Cryptography: From Practice to Theory

by

Harish Karthikeyan

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

September, 2022

Dr. Yevgeniy Dodis

Acknowledgements

The path to my Ph.D. has been paved by the support and wishes of many people. This is an inexhaustive list of those to whom I owe a debt of gratitude. My advisor Yevgeniy Dodis has been a patient, inspiring, and encouraging mentor. His fondness for aesthetics and his penchant for abstraction always leaves me in awe. To the other readers of my Thesis - Marshall Ball and Allison Bishop, my many thanks for taking the time to help me with refining my rambles. To the auditors of my Thesis - Oded Regev and Victor Shoup, I am grateful for supporting right from the start of my Ph.D. journey through my qualifying examination and my thesis proposal.

My time at J.P. Morgan AI Research provided me with a unique perspective on marrying theoretic research with practical problems and my thanks to Antigoni Polychroniadou for the support and encouragement during the internship. A dissertation is never written in silo, and this is no exception. My thanks to all my collaborators for letting me a part of their work, and always leaving me inspired to pursue research. This acknowledgment would be incomplete if I did not mention the staff of the Computer Science Department, especially Santiago Pizzini and Rosemary Amico for their untiring work behind the scenes.

To my roommate and partner Robert Crowe, my many thanks for making housing in NYC affordable and being there through times - both good and bad. My sincerest appreciation to my good Judy, Vineel Kondiboyina for being my agony aunt throughout this journey. My gratitude goes to the folks at NYU Crypto Group (past and present) – Alex, Arasu, Daniel, Eli, Paul, Peter, Sandro – for their company, support, and encouragement through many a happy hour at Peculier.

Finally, to my parents, my eternal gratitude for defying that “Asian parent stereotype”, for the most part, and encouraging me to pursue whatever made me happy.

P.S. Thank you Sid for putting together this LaTeX template and making the process that much simpler.

Abstract

This work is yet another attempt to turn an age-old adage on its head by deriving inspiration for theoretical research from problems that are germane to practitioners and real-world deployment. This could be viewed as a departure from the practice of creating real-world solutions that trace their origin to theoretical research, or *ex post facto* theoretical analyses of practically deployed solutions that can be rather *ad hoc*. Specifically, we look at four different problems that are relevant for practical deployment - random number generation, searching over encrypted data, and forward-secure group messaging.

Randomness Generation The bedrock of cryptography remains high-quality randomness that helps secure our various applications. This is achieved, in practice, by using a primitive called a *pseudorandom number generator with input* (PRNG). The inputs to the PRNGs are sourced from the physical environment using measures such as temperature and timing of interrupts. PRNGs convert these inputs into outputs that are uniform, independent, and fresh. It is important to note that most of these theoretical constructions are *seeded*. In other words, they require a random seed to be chosen at setup, and the inputs are required to be independent of this seed. It has also been shown that these constructions fail if this independence is lost. Unsurprisingly, despite being borne out by theoretical rigor, these constructions have found few takers in the real world. On the other hand, constructions in practice are ubiquitous but lack sufficient theoretical justifications. Each operating system comes equipped with its PRNG software e.g., `/dev/random` for Linux,

Yarrow for MacOS, and Fortuna for Windows and these critically rely on cryptographic hash functions without any reasonable justification, yet they seem "robust". We confront the question in this thesis - Can we capture this belief, and theoretically justify the construction choices?

Another closely allied problem is that of PRNGs that are robust against *premature-next* attacks (Kelsey *et al.*, FSE'98). In these attacks, an adversary requires the PRNG to produce outputs before sufficient entropy has been accumulated. To mitigate these attacks there exists two broad solutions - somehow estimate entropy (deployed in Linux's `/dev/random`) or use sophisticated pool-based approaches as in Yarrow and Fortuna. Hitherto, the only theoretical study in this class of PRNGs is the work of Dodis et al. (Algorithmica '17) focused on the seeded setting of this problem or assumed that the entropy arrived at a constant rate. This considers the seedless variant of this class of PRNGs.

Formally, the contributions of this thesis in this domain are as follows:

- We put forth new definitions of robustness that enables both *seedless* PRNGs and *primitive-dependent* adversarial sources of entropy. However, to bypass existing impossibility results, we model a compromise by requiring the adversary to provide sufficient entropy, yet permitting the adversary to evaluate the underlying primitive.

- We also present provably-secure constructions based on hash-function designs where the underlying primitive are ideal compressing functions, block ciphers, and permutations. These are natural and practical constructions that can be instantiated with minimal changes to industry-standard hash functions SHA-2 and SHA-3, or HMAC (as used for the key derivation function HKDF).

- Of independent interest, our constructions can be downgraded to *(online) seedless randomness extractors*. We also show that an extractor based on CBC (which forms the basis of Intel's on-chip RNG) is insecure in our model.

- Along the way, we also present two different variants of "robustness" - computational

(where an adversary makes a bounded number of queries to the ideal primitive) and information-theoretic (where an adversary makes a bounded number of queries at the beginning, but then there are no limitations after the challenge).

- We have an impossibility result that rules out the existence of seedless PRNGs that achieve security against premature-next attacks, even in a weak model (i.e., the entropic inputs to the PRNGs are independent). Note that in this model, there exist seedless PRNGs that are robust when premature-next attacks are not considered.

- In the face of the previous impossibility result, we take a closer look at the pool-based approach and we show that (a) under natural conditions (informally, the entropy cannot vary "wildly" in a single round-robin round) on the entropic input, we can prove the security of the round-robin entropy accumulating PRNGs such as Fortuna, and (b) the "root pool" approach (used in Windows 10) is secure for general entropy inputs, provided that the state of the PRNG is not compromised after startup.

Small-Box Cryptography. A common recipe to build block ciphers seems to involve a small cryptographic component (called an S-box) and then iterating this small component across several rounds. Unfortunately, there does not exist a rigorous theoretical framework to argue why this yields a secure construction and it is a longstanding open problem. A key issue lies with the tools in our arsenal. We typically show that the probability that construction is insecure is negligible (typically 2^{-n} where n is the size of the component. Indeed, prior approaches to proof of security – which we dub "big-box cryptography" – always allowed for n to be as big as the security parameter (and note that typical S-boxes have very small n). As a direct consequence, we obtained constructions that did not match any of the specifications of the practical constructions. Oftentimes, the big-box approach abstracts away the key component that brings hardness to the table.

In this work, we introduce a novel paradigm called "small-box cryptography" that provides a robust framework to justify the security of existing block ciphers. This paradigm, unlike the "big-box" approach, allows diving deep into existing constructions by idealizing the building block of size n which is small (e.g., 8-to-32-bit S-box). Through a sequence of clean, rigorous proofs and hardness conjectures, "small-box cryptography" enables security justifications for actual constructions in the real world. We get reasonable hardness estimates by applying our framework to the analysis of substitution- permutation network (SPN) based ciphers. Further, we also apply the framework to designing pseudorandom generators (aka stream ciphers). Our approach yields a simple construction that can be proved secure in the "big-box"-framework, under a well-studied and widely believed eXact Linear Parity with Noise (XLPN) assumption.

Encrypted Search We find ourselves increasingly storing sensitive documents on the cloud, where we wish to protect these documents from the wandering eyes of the cloud server. The simple solution to this problem is to encrypt these files and then store them on the server. The server, without knowledge of the secret information, is unable to decrypt these files. However, this also means that the server is unable to search (or perform other operations) on these encrypted documents which leaves the client with the unfortunate job of downloading the encrypted file, decrypting the downloaded file, and then searching over the decrypted contents. Oftentimes, we outsource the storage of these files because of their size, and therefore, doing this process repeatedly is quite expensive. This led to the birth of research on searchable encryption where one stores an encrypted index E, along with the encrypted file to help simplify the search process. Now, if we wanted to search for a keyword w in a particular document D, the sever is provided with an "encrypted token" z_w (which does not reveal information about w) to lookup z_w in the encrypted index E and simply return the result of this lookup instead.

In this work, we consider the following motivating setting: index-creator(s), using some publicly available information PK of the search approver, create an encrypted index E for a document

D. This index E is stored with the storage manager. Later, the storage manager is asked to determine if a keyword w is present in the underlying document. The search approver, approves this search request, by using the secret information SK to generate a token z_w, *that is specific to* D *and* w. The storage manager uses z_w to search in E to determine if the keyword w is present in document D. We build the first *sub-linear* (possibly constant-time), *public-key* searchable encryption scheme in this setting where:

- the storage manager, with knowledge of z_w, is unaware if $w' \neq w$ is present in document D or if w is present in document $D' \neq D$,
- the search approver does not learn any information about D, beyond the keyword w that is being searched, and
- the search approver cannot generate a malicious token z_w that forces the storage manager to print the incorrect result.

We call this primitive *Encapsulated Search Index (ESI).* Our ESI scheme can be extended to (t, n)-distributed among n different search approvers such that the scheme is non-interactive, verifiable, and resilient to any collusion among $(t-1)$ malicious search approvers. We also introduce the notion of delegation where a search approver can delegate the approval to another user without having to recompute the index. Our solution – which includes public indexing, sub-linear search, delegation, and distributed token generation – is a commercially available product created by Atakama.

Updatable Public Key Encryption Forward security (FS) requires that corrupting the current secret key of the system does not compromise any prior utilization of the system. In the context of public-key encryption (PKE), this is achieved by dividing time into smaller periods or epochs where a new secret key is derived from the current secret key while keeping the public key unchanged. The current realization of FS-PKE requires one to use hierarchical-identity-based

encryption and is rather complicated. Recent work by Jost *et al.*(Eurocrypt'19) and Alwen *et al.*(CRYPTO'20), motivated by applications to secure group messaging, considered a weakening of FS-PKE which they called *updatable* PKE. Specifically, a sender initiates a transition to the next period by generating an *update ciphertext* and a new public key. The receiver, on the input of the update ciphertext, can compute the new secret key which is consistent with the new public key. Critically, the sender does not know the original secret key of the receiver. Their construction is provably secure in the random oracle model.

In this dissertation, we introduce the first constructions of the primitive in the standard model under the DDH Assumption and LWE Assumption, the latter of which is believed to be post-quantum secure. We take a modular approach to building our schemes by leveraging three key properties of the underlying PKE scheme - circular security and leakage resilience, key homomorphism, and message homomorphism. Indeed, our constructions are much more efficient than current FS-PKE schemes built from the *same assumptions*.

Contents

List of Figures

List of Tables

1 Introduction

A standard paradigm of pedagogy is to translate theoretical framework and results into practically feasible solutions. Indeed, this has proven fruitful in cryptography, where several deployed protocols were initially born from theoretical research. Yet, there is undeniable a yawning abyss between theoretical cryptography and applied cryptography. This dissertation aims to go from practice to theory, whereby we study problems that practitioners confront through a rigorous theoretical framework. This dissertation covers three problems that originate from practice, intending to look at solutions to these problems through rigorous theoretical analyses.

The results presented in this dissertation are based on works by Coretti *et al.* [CDKT19a], Coretti *et al.* [CDK+22], Dodis *et al.* [DKW22a], Dodis *et al.* [DKW21], and Aronesty *et al.* [ACD+22a]. Passages are taken verbatim from these works throughout this dissertation.

1.1 Pseudorandom Number Generation

High-quality randomness is a critical requirement for the security of all commercially deployed cryptographic protocols. News sources are constantly rife with details on the failure of protocols stemming from poor randomness and choices made in this regard. To solve the practical problem of generating randomness, we use the primitive known as *pseudorandom number generator with input* (PRNG). Informally, they accumulate entropy from various sources in the environment, such as keyboard presses, the timing of interrupts, temperature sensors, etc., into the state

of the PRNG. Then, one can extract pseudorandom bits from this high entropic state. All operating systems come equipped with their version of PRNGs, e.g., `/dev/random` [Wik21] for Linux, Yarrow [KSF99] for MacOs/iOS/FreeBSD, and Fortuna [FS03] for Windows [Fer13]. Yet, most security analyses of these implementations tend to be after-the-fact of commercial deployment and often throw up glaring issues. For example, Dodis *et al.*[DPR+13] showed that `/dev/random` is not robust. Such work reiterates the need for a substantial theoretical understanding of these PRNGs to ensure their continued usage is secure, avoiding undesirable surprises.

On the other hand, considering the "simplistic" nature of PRNGs and their heavy usage in other cryptographic protocols, it is surprising that there has not been sufficient breakthrough in the standardization of PRNGs [ISO11, Kil11, ESC05, BK12b] when compared with some of the other primitives (e.g., such as block ciphers [DR02], hash functions [SR12, Div14], stream ciphers [RB08], public-key encryption [Sho01], digital signatures [KSD13], and authenticated encryption [AFL16]). Indeed, it is prudent to note that standardized cryptographic PRNG constructions are also proposed in an ad-hoc fashion and have been broken later by the collective analyses of the cryptographic community. The most infamous example is the DualEC PRNG which was a part of the initial draft of the NIST SP 800-90A standard [BK12b], back in 2005. Despite repeated warnings by researchers [SS06, SF07] and its potential for exploitation [CNE+14], it was finally deprecated by the actions of a whistleblower. Most recently, Woodage and Shumow [WS19] analyzed the standard with greater care and identified gaps and imprecisions in the remaining three recommendations. The earlier work of Shrimpton and Terashima [ST15] identified lapses in previous analyses, and security proofs for the popular Intel Secure Key Hardware PRNG introduced in 2011.

The above discussion leads us to our goal of bridging theory and practice on this critical piece of the cryptography ecosystem, where we wish to design a rigorous, theoretically sound model of PRNGs that are of immense interest to the practitioners. The model should be permissive to allow for real-life entropy sources while also restrictive enough to prove the security of the

constructions formally.

1.1.1 Existing Theoretical Models

A rich body of literature has focused on a formal, theoretical study of PRNGs [BH05, DPR+13, ST15, DSSW14, GT16, Hut16a]. They differ in details, but for this work, we can look at the critical unifying principles of these works. On the one hand, they require security against possibly adversarial entropy sources, provided the adversary infuses the PRNG with sufficient entropy. On the other hand, they wish to minimize assumptions on the structure of these sources beyond entropic requirements. Another point to note is that all of these constructions are *seeded*, i.e., a seed is somehow randomly chosen at initialization, and the entropy sources are assumed to be *independent of this seed*. This assumption was an artifact of the modeling from prior work on randomness extraction, [NZ96] which showed that seeded extractors bypass the impossibility result of deterministic (i.e., "seedless") randomness extraction from general entropy sources [CG88].

However, in the context of PRNGs, such modeling seems inherently problematic. To begin with, one uses PRNG to produce high-quality random bits. Yet, the initialization procedure requires the seed to be chosen randomly, leading to the circular problem of needing random bits to produce. A simple solution to this problem might be to rely on other methodologies, apart from PRNGs, to produce this one-time, high-quality random seed. However, the issue of requiring that the seed be independent of the entropy sources persists. Indeed, it has been shown that if the inputs were somehow dependent on the seed, the PRNGs would catastrophically fail [DPR+13]. The immediate question is on the likelihood of this event. Unfortunately, it is quite a realistic scenario. For example, if one were to consider physical events in the computer as the entropy sources, then an execution of a PRNG operation to produce pseudorandom outputs would trigger interrupts or other such physical events. Thus, executing the PRNG using the seed triggers events that can become subsequent inputs to the PRNG. This loss of independence is also of concern in the context of "premature next", where the adversary can request outputs from the PRNG, even

when it has failed to provide entropy adequately. In such a scenario, the adversary can recover the seed leading to the loss of independence.

This discussion leads us to the following dilemma. To support general entropy sources, one needs to resort to seeded extraction, as seedless extraction is impossible for such sources. Yet, it is evident that one has to concede the cons of seeding a PRNG as seeded extraction is only possible due to this hard-to-realize independence assumption. Through this work, we hope to provide a meaningful solution to this dilemma.

Seeded PRNGs and Premature Next Attack. We clearly understand how a PRNG should behave when "sufficient entropy" has been provided. The expectations are that the outputs are pseudorandom. However, one can consider the situation where the PRNG is expected to produce pseudorandom outputs before the system has accumulated enough entropy. This situation can occur at system startup, where we want randomness immediately after startup. One can conceivably mount an attack where an adversary prematurely requests outputs from the PRNG and then uses the received output to invert the computation and compromise the security of the PRNG. Premature Next problem was first proposed as an attack by Kelsey *et al.* [KSWH98]. Since then, the only work that solidified the notion of a "robust" PRNG secure against premature-next attacks was done by Dodis *et al.* [DSSW17].

1.1.2 Our Approach

In this work, we use cryptographic hash functions as the technical tool, carefully define the notion of entropy, and leverage some idealized computation models.

Cryptographic Hashing. It is essential to note that all general-purpose software PRNGs and existing recommendations in PRNG standards critically rely on cryptographic hash functions (CHF), despite lacking any substantial theoretical justification. One can also provide a more the-

oretical reason why a CHF might be a natural fit for PRNG construction. Recall that the accumulation function of PRNG (denoted by refresh in the syntax) takes a PRNG state S, an entropic input X, and accumulates the entropy into a new state S' where S', ideally, should have larger entropy than S. This property is commonly referred to as "condensing" in extraction literature, and Dodis *et al.* [DRV12] showed that seedless/seed-dependent condensers could only be built cryptographically using properties of preimage-resistance and collision-resistance. Indeed, even setting aside a desire for seedless PRNGs (or the more straightforward case of seedless extraction), the simple property of seedless entropy accumulation appears to require the use of CHF, along with a supporting justification argument in an idealized model of computation.

Idealized Model of Computation. Consider the more straightforward problem of monolithic seedless extraction using a cryptographic hash function G. We will model this G as an ideal primitive – random oracle – and set $R = G(X)$. Now, note that we have a trivial construction of a seedless extractor by the following folklore proof (see [DGH+04]): For any source, X with min-entropy γ^*, the probability that the distinguisher $\mathcal{D}$ can distinguish $G(X)$ from a truly random value is bounded by $q \cdot 2^{-\gamma^*}$ where q is the number of queries that $\mathcal{D}$ makes to the random oracle G. However, we have implicitly used the fact that the source X was neither adversarially chosen adversary nor did it depend on the random oracle. However, to model reasonable sources arising from practical implementations, one needs to allow for oracle-dependent sources, i.e., where the adversary can query the underlying idealized primitive before providing the input.

Min-Entropy Notion. This brings us to the question of how to allow for oracle-dependent sources reasonably. The naive way that one requires input X to have sufficient entropy is not restrictive enough, for we have the following attack: an adversary samples X at random, writes it down in its state Σ and then provides that as input. X has full entropy, but with Σ containing X, $G(X)$ is no longer pseudorandom. The tempting fix is to condition it on Σ, i.e., require that $\mathrm{H}_\infty(X|\Sigma) \geq \gamma^*$. Unfortunately, it still leaves us open for an attack that we call *extractor-fixing*

attack. An adversary merely samples X several times until we have that the first bit of X is 0. Now, X has very high entropy (almost full entropy), but $G(X)$ is not pseudorandom. Note that the condition is met even if one requires that $\mathrm{H}_\infty(X|\Sigma, G) \geq \gamma^*$. Instead, we simply condition only on the queries made by the adversary to the random oracle, i.e.,

$$\mathrm{H}_\infty(X|(\text{state}, \mathcal{L})) \geqslant \gamma^* , \tag{1.1}$$

where $\mathcal{L}$ is the *input-output* list of random oracle queries made by the sampler $\mathcal{A}$ to the random oracle. It is important to stress that our modeling also captures the case when the input X is independent of the oracle by merely setting $\mathcal{L} = \emptyset$. Finally, our legitimacy condition is much stronger than the definitions considered by Dodis *et al.* [DPR+13] and Gazi and Tessaro [GT16]. These works employed the weaker notion of worst-case min-entropy. They defined the final entropy of the source X as the total of worst-case conditional min-entropies of individual blocks of X where the conditioning was on the remaining blocks.

1.1.3 Our Results

The simpler setting of monolithic seedless extraction was a useful toy example to understand and motivate our new definitions. However, access to such a monolithic random oracle is hard to realize. Thus, we find most cryptographic hash functions built from (public) compression functions, ciphers, or permutations as the underlying primitive P, which have limited input length. The standard approach is to process an arbitrarily large input of size m in smaller n-bit input blocks and then accumulate entropy.

- In Section 3.4, we present the more realistic notion of seedless, online extraction. Such extractors accumulate entropy by processing smaller input blocks of length m, using an ideal primitive P, and then produce an output once the entire input is processed. We model security games for both the computational and information-theoretic versions.

- In Section 3.5.5, we show that the CBC mode of operation is insecure as a seedless extractor in our framework. This result is critical because CBC is used as the extractor that underpins the CTR_DRBG construction of NIST SP 800-90A Rev. 1 [BK12a], and also as the extractor for Intel's on-chip RNG [M14]. Independently, its security was proved by Dodis *et al.* [DGH+04] when the input was assumed to be independent of the ideal primitive.

- In Section 3.5, we show several feasible online extractors based on SHA-2 and SHA-3 and prove their security. Given our positive results, it is to be contrasted with the negative result for the CBC mode of operation. Our positive results build extractors from Merkle-Damgård using a random compression function, Merkle-Damgård with Davies-Meyer compression function, and from Sponges.

- Our new definitions of robustness enable seedless PRNGs in an idealized model of computation while allowing for entropy sources that can be dependent on the underlying primitive and can be adversarial. This modeling is done to bypass the famous impossibility result and capture a realistic model. To this end, we define a new legitimacy condition that requires that the adversary provide sufficient entropy, conditioned on its evaluations of the underlying primitive. The reader can find these definitions in Section 4.1. The definitions include two variants of "robustness" - computational and information-theoretic. The former restricts the computational power of the adversary before and after the challenge, while the latter only restricts the computational power before the challenge. Finally, we include a comprehensive discussion about how the various definitions in literature compare with our new definition of robustness in Section 4.4.

- We combine our discussions from seedless extraction and the new robustness definition to present our provably-secure constructions for fully seedless PRNGs in Chapter 9. These constructions are built as Merkle-Damgård using a random compression function, Merkle-Damgård with Davies-Meyer compression function, and from Sponges.

- In this work, we also present an impossibility result that rules out the existence of seedless PRNGs that achieve security against premature-next attacks, even in a weak model (i.e., the entropic inputs to the PRNGs are independent). The attack critically leverages the arrival of entropy at a languid pace. Therefore, it is natural to look at feasibility results where we impose conditions on the arrival of this entropy. Our modeling follows the parametrization of security by γ^* and β, similar to [DSSW17]. If the PRNG has received γ^* entropy in T^* steps since the last state compromise, we expect recovery from compromise within $\beta \cdot T^*$ steps. We show that for any choice of γ^*, β, an efficient adversary exists providing $q \geq \gamma^{*2}\beta^2$ PRNG inputs (each with one independent bit of entropy), which violate the PRNG security against premature next attacks. Since q is typically huge, this rules out any reasonable settings of γ^* and β.

- It is important to note that we have practical solutions that seem to work against premature-next calls. On the one hand, we have the pool-based, round-robin approach adopted by Yarrow [KSF99] and Fortuna [FS03]. On the other, we have Windows 10 [Fer19] which uses a special "root pool" to solve the problem of initial entropy accumulation when the computer starts up. This single pool is emptied at exponentially increasing intervals (e.g., at time 1, β, β^2, ...) to (heuristically) solve the problem that sometimes the computer might boot with no good source of randomness for an unknown period. Therefore, we ask ourselves what meaningful security relaxation would help us prove the security of these constructions, despite the impossibility result even if one were to assume perfect entropy accumulation and extraction. To this end, we define a clean model of seedless (pool-based) *schedulers*, extending the corresponding notion of schedulers [DSSW17] to the seedless setting in Section 6.3. Intuitively, if we have k pools, given each entropic sample X_i, the scheduler decides which pool $\mathsf{in}_i \in [k]$ will accumulate this entropy, and, which pool $\mathsf{out}_i \in [k]$ (if any) will contribute its accumulated entropy back to the register. Moreover, to model ideal entropy accumulation and extraction, we assume that the entropy that is thrown to pool i simply

adds up without loss.

- We show in Section 6.4 that the root-pool approach achieves nearly optimal (α, β)-security to accumulate entropy at start-up, where $\alpha \approx \log_\beta q$ (and we can take any integer $\beta \geq 2$). Plugging in known constructions yields a PRNG in the root-pool model (i.e., in which we assume that compromise only happens at time 0) that is exponentially better than our general-scheduler lower bound stating $\alpha\beta \geq \sqrt{q}$.

- Finally, in Section 6.5, we show that the round-robin Fortuna construction with $k \geqslant \log_\beta q$ pools achieves (α, β)-security with $\alpha \approx \log_\beta q$, provided one uses a more conservative notion of entropy called k-smooth entropy.

1.1.4 Related Work

We mention some important categories of related works, in particular concerning seedless extraction, PRNGs, and their security.

Seeded and Seedless PRNGs. Extending the prior definition of a monolithic PRNG by Barak and Halevi [BH05], Dodis *et al.* [DPR+13] introduced the definition of a robust PRNG and presented an extensive theoretical foundation hitherto missing for this primitive. This work has been extended by [DSSW14, GT16, Hut16b], with the latter two works being analyzed in the random permutation model (in addition to the seed). However, none of these works considered a seedless setting for general entropy sources. Inspired by the work of Coretti *et al.*[CDKT19a], which forms a part of this work, there has been additional research on constructions of Seedless PRNGs. Subsequent works by Dodis *et al.* [DGSX21b, DGSX21a] considered the simple case where the inputs are *independent* without assuming ideal primitives.

Extractors and PRNGs in ideal models. Ideal model analysis of extractors and PRNGs was done by several works [DGH+04, BDPV10, ST15, WS19]. Despite the constructions being explicitly seedless, their work assumed that the *entropy source was independent of the ideal primitive.*

Note that goal of these papers was not to design theoretically optimal extractors or PRNGs but to engage in an after-the-fact analysis of the heuristic use of various cryptographic hash functions and popular modes of operations (such as CBC, HMAC, etc.) for randomness generation and extraction.

Extraction from Structured Sources. Another popular approach to bypassing the impossibility result was to assume that the entropy source X was more structured beyond entropy (e.g., various bit-fixing, limited dependence sources, from several independent sources)[von51, CGH+85, Blu86, LLS89, CG85, BIW04, CZ16]. While theoretically elegant and exciting, these are not amenable for usage in realistic PRNGs. Taking another approach, Barak *et al.* [BST03] allowed for arbitrary min-entropy sources but assumed that these were sourced from an a-priori bounded number of distributions. Unfortunately, their construction does not appear helpful for general-purpose PRNGs. On the one hand, their work focused on "monolithic" extraction, which, as we argue above, seems to lack a practical basis. On the other, they use the so-called t-wise independent hash functions, with a large choice of t making it inefficient.

Low-Complexity Samplers. Unlike this work where the entropy sources can run the extractor, with the definition of legitimacy ruling out the extractor-fixing attack, low-complexity samplers assume that the entropy source cannot run the extractor even once. Low-Complexity Samplers were introduced by Trevisan and Vadhan [TV00] and later extended by [KRVZ11]. While interesting in situations where the entropy source is straightforward, we believe that this assumption is unrealistic for general-purpose PRNGs considered in this work.

Randomness condensers. This approach, formalized by Dodis, Ristenpart and Vadhan [DRV12], relaxes the security guarantees of the randomness extractor to only ensure that the output of the (seedless or "source-dependent-on-seed") condenser is almost full entropy, despite not being perfectly uniform. Indeed, this weaker security turns out to be sufficient for several applications, such as key-derivation schemes for signature schemes. Unfortunately, if we want an extractor

rather than a condenser—which is essential for general-purpose PRNGs—this approach is not sufficient.

UCEs and public-seed pseudorandomness. The notion of universal computational extractors (UCEs) [BHK13], and its generalizations [ST17], study a complementary problem to the one studied here: how to extract from an entropy source which is only *computationally*-hard-to-predict, so it only has "computational entropy". On a positive, and similar to this work, when instantiated with constructions from an ideal primitive P, a UCE hash function yields a good extractor even if the inputs to it (the actual source) can be sampled depending on the ideal primitive. The issue, however, is that the current UCE notion inherently *requires a seed*, making it inapplicable for the PRNG scenario. An interesting direction for future research could be to extend our work to deal with computational entropy by defining and constructing *seedless* UCEs in idealized models, and possibly extending them to full-blown seedless PRNGs for computational entropy.

1.2 Small-Box Cryptography

Block ciphers are workhorses of cryptography and are used everywhere. There are a large number of candidate constructions for this primitive, including the immensely popular and widely used AES. The key idea of these constructions involves repeating a sequence of invertible transformations over several iterations. There are two approaches to building these block ciphers: Feistel networks [Fei73] or substitution-permutation networks (SPNs) [Fei73, Sha49]. Feistel networks begin with a keyed pseudorandom function on n-bit inputs and extend this to give a keyed pseudorandom permutation on $2n$-bit inputs. SPNs start with one or more public "random permutations" on n-bit inputs and extend them to give a keyed pseudorandom permutation on wn-bit inputs for some w.

1.2.1 Big-Box Cryptography

Ideally, our preference would be for an unconditional justification for this approach of achieving hardness by iterating a "simple component" over multiple rounds. Unfortunately, such a proof is not immediate and would certainly imply that $P \neq NP$, solving the famous problem. Instead, results thus far [MV15] have focused on specific (and limited) security properties of block ciphers such as their resistance to specific types of attacks (e.g. linear, differential, etc.) and these are insufficient to prove the security of real-world constructions of authentication and encryption that extensively rely on these block ciphers. For example, we need to prove their resistance to CPA/CCA style attacks and security proofs towards this purpose tend to follow a three-step framework:

1. *Abstraction:* a building block f inside a single round of the corresponding cipher is abstracted away and idealized.

2. *Proof:* formally demonstrate the security of this abstracted and idealized block cipher, for some minimal number of rounds r, by using standard reduction techniques.

3. *Conjecture:* then conjecture that for real-world applications where we have the iteration over some $r' > r$ where r was calculated in the previous step ensures its security. The conjecture part of this argument stems from the fact that the actual construction of this building block f is typically far from ideal behavior.

This three-step approach is what we dub as *big-box* cryptography. The first issue one notices is that the idealization requires the block f, of size n, to be proportional to the length N of the block cipher. The seminal paper by Luby and Rackoff [LR88] showed that a 4-round Feistel network yields a secure pseudorandom permutation (PRP) for $N = 2n$ bits when we apply the iteration on n-bit round functions which are independently keyed and modeled as a pseudorandom function. For the context of SPN ciphers, security poof approaches have abstracted away the substitution-

permutation structure and viewed the constructions as *key-alternating ciphers* [EM93, BKL+12, CS14, HT16]. In other words, the entire SPN layer is replaced by a monolithic public permutation Π on $N = n$ bits. This coarse abstraction yields a result that a r-round key-alternating cipher is secure for any $r \geq 1$, in the random permutation model on N bits [EM93, BKL+12, CS14, HT16]. $r = 1$ is the setting of the famous Evan-Mansour cipher [EM93]. In the end, our results in the big-box world have the flavor of exact security bounds that are exponentially small in the block length $N = O(n)$ of the cipher and reduce the security of the cipher to this smaller and simpler idealized building block f.

Unfortunately, the coarseness of the abstraction allows for the more interesting (at least in our opinion) parts of the problem to be lost or ignored. For example, the "meat" of the problem is building this block f and if one needs f to be "big" (hence the term "big-box"), it does not convey anything to the practitioners on how to build this block f in real life for no real-life implementation would meet the expectations of ideal behavior, especially if f is expected to be small in real life. Indeed, the round function of DES or any other Feistel cipher is far from pseudorandom and the same is true for a 1-round SPN structure being nowhere near the expected behavior of a PRP. Indeed, the theoretical rounds required are seldom sufficient for secure deployments which is why we do not have a 1-round SPN cipher or a 4-round Feistel cipher. Further, a poor choice of the round function can render the entire Feistel network-based block cipher insecure, and ignoring the SPN structure by reducing the analysis to a key-alternating cipher ignores the crux of existing constructions.

To summarize, big-box cryptography provides attractive bounds and can certainly serve as a sanity check. However, the abstraction is far too coarse that it does not provide any useful information for practitioners on how to actually build their construction.

1.2.2 Our Approach

We were motivated by the aforementioned issues with the standard big-box approach, and our key approach is a new framework which we name "small-box cryptography". Our framework consists of two steps:

1. *Construction Step.* This step itself consists of two components specific to the primitive (e.g., block cipher, hash function, etc) we are building: *domain extension* and *hardness amplification*. Despite being primitive-specific, it is largely *syntactic*, resulting in many constructions that have the potential to be secure in the real world.

2. *Analysis Step.* This step gives concrete exact security bounds/conjectures for the resulting constructions. It consists of three parts. The first two parts are *information-theoretic* and *fully provable*.[1] They formally analyze the domain extension and hardness amplification steps above within the existing techniques from "big-box" cryptography. The last step introduces a new "big-to-small" conjecture, which allows one to lift these big-box results to meaningful bounds/conjectures about the security of the resulting construction in the real world. In essence (see Theorem 7.14), this conjecture states, that if a natural-looking hardness amplification result gave a good security $\varepsilon(n)$ against attackers running in time T assuming n, is "large" ($n \gg \log T$, in particular), then the same construction will also have security $\varepsilon'(n) \approx \varepsilon(n)$ even for much smaller values of n, although the supporting security proof critically breaks down in this case.[2]

[1] In practice, the hardness amplification step is often used with correlated round keys, using some "key schedule" heuristic. To model this case, we also need a plausible conjecture that the key schedule step does not violate the information-theoretic security proven using fully independent round keys.

[2] As we will see, the "big-to-small" conjecture looks very different from all previous ("big-box") hardness assumptions, and could be viewed as "one-way function" of small-box cryptography. While the particular conjectures introduced here might be too strong/aggressive or require further fine-tuning, the framework is general enough to accommodate future milder variants of this conjecture, still leading to meaningful real-world guarantees, while addressing the limitations of big-box cryptography.

Our new[3] approach attempts to go much deeper inside the existing block cipher constructions, by only idealizing a small (and, hence, realistic!) building block f, such as an S-box. For example, let us recall that an SPN cipher on wn bit inputs (where w is a relatively large constant $w \geqslant 1$), is computed via repeated invocation of two basic steps: a *substitution step* in which a public (unkeyed) "cryptographic" permutation $f : \{0,1\}^n \rightarrow \{0,1\}^n$, called an *S-box*, is computed in a blockwise fashion over the wn-bit intermediate state, and a *permutation step* in which a keyed but "non-cryptographic" permutation π on $\{0,1\}^{wn}$ is applied, called a *D-box*. Since π is non-cryptographic and typically linear, we will not idealize any of its properties, and work with D-box permutations π close to those used in practice. Hence, the only component which can be idealized is the S-box f, which we will model as a random permutation. Since the input length, n of f is small, such idealization is not unreasonable, which means the final construction analyzed is *close* to what is used in practice, and certainly captures the heart of the SPN construction: namely, the *actual SPN structure*, as opposed to key-alternating ciphers, where this structure is completely ignored!

Of course, given the huge conceptual advantages of the small-box approach over the big-box approach in terms of the "abstraction" step, there is an important catch, as otherwise, we would likely have an unconditional result (and proved $P \neq NP$ along the way). The catch is that the best provable security one can conceivably get with such an approach is only exponential in n, as the S-box was the only idealized source of hardness that we could use. And since $n \ll N$ was very small by design (say, at most 32 in existing constructions), the actual bounds are not useful for practical use. At first, this admittedly serious deficiency appears to invalidate the whole point of provable security with this approach, which might have been the reason why so few papers followed this route before this work. However,

As one of the contributions of this work, we show that the seemingly useless bounds one gets in the

[3] As we detail in the related work Section 1.2.4, some of our ideas were already used in the prior work, but not in the totality that we present here.

"proof" component of the "small-box" approach, can still lead to very reasonable final results,

provided one properly models the "conjecture" component of this approach.

The approach is rather subtle and is carefully explained in Section 7.3. In brief, it formalizes two clean and explicit hardness conjectures, termed *hardness amplification* (Conjecture 7.13) and *big-to-small* (Conjecture 7.14). The hardness amplification conjecture, which is very plausible and can be sometimes proven even *unconditionally* (under appropriate independence assumptions) using a beautiful hardness amplification result of Maurer, Pietrzak, and Renner [MPR07], states that the success probability ε of the distinguisher can be driven down exponentially by cascading the block cipher with itself.[4] Notice, that such cascading is indeed a common practice of every block cipher design, where increasing the number of rounds (with independent or even correlated keys) is critical for improving the security of the block cipher. In particular, we can get this success probability to an extremely low level of 2^{-wn} by cascading the original cipher $O(w)$ times.

However, this conjecture is only meaningful in the "big-box" setting, when the size n of our building block (e.g., S-box) is larger than the security parameter, as otherwise the exponential in n bounds given by our "proof component" are meaningless. To go back to the small-box case we care about, we notice that the success probability 2^{-wn} achieved in the big-box setting after cascading is also good and meaningful in the small-box case. The big-to-small conjecture states that even though the hardness amplification argument used to justify this conclusion crucially relied on the big-box assumption, the *conclusion is true even in the small-box case*! Unlike the hardness amplification step, which appears very believable and even unconditionally true in certain settings, the big-to-small conjecture is completely new and not formally studied. However, despite being new and rather strong, it allows us to precisely state the kind of "leap of faith" one would be making when using constant size small-boxes.

We discuss these issues in more detail in Section 7.3, here only stating the result of applying

[4] While we state this result for block ciphers, the framework of [MPR07] is strong enough to study unconditional hardness amplification for other primitives, such as PRGs (where one XORs several PRGs with independent seeds).

the 2 conjectures together. Here $n_0 = n_0(a, \alpha)$ is the constant defined in the big-to-small conjecture (and could be small; $n_0 = 8$ in the case of AES), and we also don't explicitly state if cascading uses independent or correlated keys/building blocks (which is part of the hardness amplification conjecture):

Theorem 1.1 (Small-Box Cryptography; Informal)**.** *Let T be the desired attacker time bound, and assume that r-rounds block cipher E of length wn utilizing idealized block f of size n is $(T, 2^{-\alpha n})$-secure, as long as $n > a \log T$ (for some constants $a > 1$ and $\alpha < 1$). Then, under Conjectures 7.13 and 7.14, for any $n \geqslant n_0(a, \alpha)$, cascading E for $c = O(w/\alpha)$ times will result in a $r' = O(wr/\alpha)$-round block cipher E' which is $(T, O(T/2^{\ell(n)} + 2^{-wn}))$-secure,[5] where $\ell(n)$ is the key length of E' under to corresponding cascading step (equal to c times the key length of E when independent keys are used).*

The theorem above formalizes the last, "conjecture" step of small-box cryptography to get the following conclusion:

Under two clean and explicit hardness conjectures, one can get strong and meaningful security bounds for popular block ciphers, by obtaining "seeming useless" $(T, \text{poly}(T)/2^n)$ security bounds for reduced-round variants of these ciphers with idealized building blocks of size n.

Moreover, the small-box approach explicitly explains why the number of rounds r' used in practical constructions is *noticeably larger* than the theoretically predicted number of rounds r in the provably secure step: to drive the success probability of the distinguisher significantly below the minimum 2^{-n} level possible with the traditional information-theoretic proof. Thus, we have eliminated both significant disadvantages of the big-box approach: not guiding how to instantiate the "big" building blocks in practice, and giving inadequately low predictions for the number of rounds r needed for real-world security.

[5] For simplicity we consider uniform attackers; for other (e.g., non-uniform) models, we can change the conjectured $T/2^{\ell(n)}$ term to reflect the best generic attack in this model; see [CDG18] for such non-uniform bounds for block ciphers.

1.2.3 Our Results

We believe our main result is conceptual: bring the attention of the cryptographic to the deficiencies of "big-box" cryptography for the task of designing block ciphers and other symmetric key primitives, which are usually built from scratch, from very small components such as S-boxes. We also introduced a specific framework (which we called *small-box cryptography*) which is one concrete attempt to address this problem. This framework yields a rather syntactic way to derive candidate constructions conjectured to be secure in the real world and then proposes an explicit way to get concrete security bounds for the resulting constructions: by combining provably secure domain extension and hardness amplification steps with a new and unstudied type of hardness assumptions we call *"big-to-small" conjectures*.

We then apply this framework to the analysis of SPN ciphers (e.g, generalizations of AES), getting quite reasonable and plausible hardness estimates for the resulting ciphers. We also apply this framework to the design of stream ciphers. Here, however, we focus on the simplicity of the resulting construction, for which we managed to find a direct "big-box"-style security justification, under a well-studied and widely believed XLPN assumption [JKPT12].

Overall, we certainly hope that our work will initiate many follow-up results in the area of small-box cryptography, which will both refine the initial heuristics (such as more refined analogs of our conjectured Theorem 1.1) outlined in this work, and add to a better understanding of existing symmetric-key constructions, hopefully well beyond block/stream ciphers.

1.2.4 Related Work

There are only a few prior papers looking at provable security of SPNs. The vast majority of such work analyzes the case of secret, key-dependent S-boxes (rather than public S-boxes as we consider here), and so we survey that work first.

SPNs with secret S-boxes. Naor and Reingold [NR99] prove security for what can be viewed as

a non-linear, 1-round SPN. Their ideas were further developed, in the context of domain extension for block ciphers (see the further discussion below), by Chakraborty and Sarkar [CS06] and Halevi [Hal07].

Iwata and Kurosawa [IK01] analyze SPNs in which the linear permutation step is based on the specific permutations used in the block cipher Serpent. They show an attack against 2-round SPNs of this form, and prove security for 3-round SPNs against non-adaptive adversaries. In addition to the fact that we consider public S-boxes, our linear SPN model considers generic linear permutations and we prove security against adaptive attackers.

Miles and Viola [MV15] study SPNs from a complexity-theoretic viewpoint. Two of their results are relevant here. First, they analyze the security of linear SPNs using S-boxes that are not necessarily injective (so the resulting keyed functions are not, in general, invertible). They show that r-round SPNs of this type (for $r \geq 2$) are secure against chosen-plaintext attacks.[6] They also analyze SPNs based on a concrete set of S-boxes, but in this case, they only show security against linear/differential attacks (a form of chosen-plaintext attack), rather than all possible attacks, and only when the number of rounds is $r = \Theta(\log n)$.

SPNs with public S-boxes. The work of Cogliati *et al.* [CDK+18] analyzed SPNs with public S-boxes. This work will give us the "domain extension" ($n \longrightarrow wn$) component of our "Analysis Step" when we apply small-box cryptography to SPNs. Unlike our work, however, the work of [CDK+18] did not advocate the hardness amplification to go beyond 2^{-n} security, or derived a concrete framework to assess the security of SPNs in the real world.

The earlier work by Dodis et al. [DSSL16] studied the *indifferentiability* [MRH04] of confusion -diffusion networks, which can be viewed as *unkeyed* SPNs.

As observed earlier, the Even-Mansour construction [EM93] of a (keyed) pseudorandom permutation from a public random permutation can be viewed as a 1-round, linear SPN in the degen-

[6] In contrast, [CDK+18] showed that 2-round, linear SPNs are not secure against a combination of chosen-plaintext and chosen-ciphertext attacks when $w \geq 2$.

erate case where $w = 1$ (i.e., no domain extension) and all-round permutations are instantiated using a simple key mixing. Security of the 1-round Even-Mansour construction against adaptive chosen-plaintext/ciphertext attacks, using independent keys for the initial and final key mixing, was shown in the original paper [EM93]. Kilian and Rogaway [KR01] and Dunkelman, Keller, and Shamir [DKS12] showed that security holds even if the keys used are the same. As we mentioned, these results are insufficient for us, as we need a much larger (at least security parameter) domain expansion factor w.

Cryptanalysis of SPNs. Researchers have also explored cryptanalytic attacks on generic SPNs [BBK14, BK, BS10, DDKL15]. These works generally consider a model of SPNs in which round permutations are secret, random (invertible) linear transformations, and S-boxes may be secret as well; this makes the attacks stronger but positive results weaker. In many cases the complexities of the attacks are exponential in n (though still faster than a brute-force search for the key), and hence do not rule out asymptotic security results. On the positive side, Biryukov et al. [BBK14] show that 2-round SPNs (of the stronger form just mentioned) are secure against some specific types of attacks, but other attacks on such schemes have been identified [DDKL15].

Hardness Amplification. Harness amplification, tracing its origin back to the seminal paper of Yao [Yao82], amplifies the security of a given cryptographic primitive, typically by combining c independent copies of these primitives, and ensuring that the attacker must break all such copies. Traditionally, it is studied in the *computational setting* (e.g. [CRS+07, CHS05, DIJK09, CLLY10, DNR04, Gol95]), where one starts with (T, ε)-security, and gets (T', ε')-security, where $\varepsilon' \approx \varepsilon^c$. Unfortunately, such complexity-theoretic results, while extremely beautiful, have an inherent limitation that $T' \leqslant T\varepsilon' \approx T\varepsilon^c$. This means that the increased security comes at the price of a huge degradation in the run-time of the attacker, making these beautiful results completely useless for small-box cryptography. See [DJMW12] for more discussion.

Fortunately, hardness amplification has also been studied in the information-theoretic setting [MPR07, Tes11], where the attacker is computationally unbounded but has a limited number

of queries T to appropriate idealized oracles. In this setting, the security can be proven without much degradation in the parameter T, and this is the setting we use in our framework.

Random Local Functions. Goldreich [Gol11] suggested designing a one-way function by repeatedly applying a certain local predicate f (which could be viewed as "S-box") to carefully chosen subsets of input bits. This influential work led to many follow-up constructions (see [App13] and references therein) of how to build various "local" cryptographic primitives in this way, and argue about their security. At a high level, these results could be viewed as a different instantiation of small-box cryptography, which is incomparable to our proposal. Namely, our proposal focuses on capturing real-world designs where security is obtained by repetition and suggests modeling f as a random function/permutation in the Analysis Step. In contrast, the study of local cryptography is more focused on achieving small input locality (which is not our concern), as a result explicitly trying to avoid naive hardness amplification (which is expensive for locality). In other words, the two approaches happen to use "S-boxes" for completely different goals. It would be interesting to see if some interesting connection can be found between the two approaches to "small-box cryptography".

1.3 Delegatable Threshold Searchable Encryption

Consider the following motivating application: Alice, working out of a shared desktop where she cannot store secret keys, wishes to store it instead on a more secure mobile phone. However, Alice working out of the Desktop generates large documents which need to be indexed separately and encrypted for storage, with the phone controlling the secure search of any words in the encrypted documents. Alice additionally wants to store preferably only one key to not waste storage on a resource-constrained device such as a phone. Indeed, our actual solution will allow several different parties to generate the index, possibly even outside of the desktop, with the assumption that the documents are available at the time of indexing, in their entirety. While the

paradigm of searchable encryption gained steam for the setting of email messages, it is evident that most documents are immutable and seldom change. One can think of patents, photos, and resumes where they are available in their entirety for the indexing operation and does not need an incremental approach to indexing.

To solve this motivating application, we introduce a new primitive known as Encapsulated Search Index (defined in Section 8.2.1) and it provides the following functionality:

- The phone generates a secret key SK and sends the public key to the desktop.
- Given PK and a document D, we can build an encrypted index E (along with an encrypted version of D). In addition, we will also have a "compact" handle c. Once E is built, the document D and any local randomness is erased. The desktop will only have PK, E, and c stored.
- Now, a user accessing the desktop wishes to check if a certain document D contains a keyword w. A trivial solution would be to decrypt the encrypted version of D and check for keyword w. However, the goal here is to make the process simpler and use the encrypted index E to search for the presence of w.
- The user approaches the phone to authorize a search for w. However, we wish to keep the communication between phone and desktop to be minimal. Therefore, rather than sending the entire encrypted index E to the phone (which could be as large as the document D), the desktop merely sends w and the compact handle c.
- The phone uses w, c, and the secret key stored to grant a search token z_w to the desktop.
- We desire privacy, i.e., that the phone does not learn anything beyond w from the handle c. No information about the index E, the randomness used, etc. This requirement is information-theoretic.

- We desire verifiability, i.e., that the desktop can verify if z_w is indeed the correct token for w. In other words, a malicious phone cannot make the desktop output an incorrect answer merely a denial of service attack.

- Post verification, the desktop can use E, c, z_w, and PK to learn if $w \in D$.

- We desire index privacy, i.e., the token z_w is specific to the pair (D, w) and therefore the desktop should not learn information on whether a word $w' \neq w$ is present in D or if w is present in another document $D' \neq D$.

- We desire efficiency, i.e., the overall search process by the desktop should be much faster than the number of keywords in D.

- We desire distribution of the trust, i.e., distribute the search approval process to multiple devices and search can only happen after we receive a quorum of approval. The resulting notion of *threshold ESI* is formalized in Section 8.2.3. This would correspond to the setting of multiple devices serving the role of the search approver.

- We desire delegation, i.e., given an index E for some key-pair (PK, SK), one should efficiently be able to delegate the search process to another user (PK', SK') without having to modify E. We formalize several flavors of such *delegatable ESI* in Section 8.2.4

- We desire updatability, i.e., update the index E by adding or deleting a keyword using just the token z_w. We call this an *updatable ESI* and is formalized in Section 8.2.5.

1.3.1 Our Approach

We begin by looking at a naive solution that will serve as a useful launchpad for our approach. In the naive solution, we combine a CCA-secure encryption scheme (Gen, Enc, Dec) and a verifiable random function VRF [MRV99, MRV99, Lys02, Dod03, DY05, GNP+15, GRPV20]. The phone

begins by sampling $(PK, SK) \leftarrow\!\!\$\ \mathsf{Gen}$, and announces PK. The desktop, to index a document D, begins by sampling a VRF key K. Then, for every keyword in w, it computes $z_w = VRF_K(w)$. Then z_w can be inserted into a non-private dictionary E. Note that each z_w is pseudorandom (and also distinct w.h.p); thus E will not reveal any information about the document D beyond the number of keywords. It also computes the compact handle as $c \leftarrow\!\!\$\ \mathsf{Enc}(PK, K)$. Desktop erases K and D, retaining only E, c, PK. To authorize a search for a keyword w, the desktop communicates c and w to the phone. The phone then decrypts c to get VRF key $K \leftarrow \mathsf{Dec}(SK, c)$ and then evaluates $VRF_K(w)$ and sends the result to the desktop. The desktop is now capable of searching the dictionary E with this value.

At first glance, this naive solution seems to work and is the solution we need. The communication sent is independent of the size of the document, and the search is efficient and sub-linear. Further, c is independent of D and therefore we have privacy, the pseudorandomness guarantees index privacy and the verifiable random function gives us verifiability. However, it does not allow us two key features that we desire - distribution of trust and delegation of search.

- For threshold ESI, achieving "decrypt-then-evaluate-VRF" functionality *non-interactively* appears quite difficult. One might be tempted to compose a one-round CCA scheme [BMW05, BBH06] with a non-interactive threshold VRF as decrypting will give shares of the key k, which you can right away turn to share of $PRF_k(x)$, maintaining one-round. Unfortunately, there appear to be only two recent non-interactive threshold VRF schemes [GLOW20]). Prior work on distributed VRFs was either interactive [Dod03, KM13], or lacked verifiability [NPR99, AMMR18] or offered no formal model/analysis [Clo, Cor20, DAO19, Kee20, SJSW19]. Unfortunately, these VRF constructions have the secret key over the standard group $\mathbb{Z}_p$, whereas the message space of the CCA encryption schemes is in the target group. Therefore, we need a new non-interactive threshold VRF with secret keys residing in the target group.

- For delegatable ESI, our definitions (and the overall application) require an efficient procedure S-Check(PK_1, c_1, PK_2, c_2) to check that the new handle c_2 was indeed delegated from c_1. The naive delegation scheme of decrypting c_1 to get VRF key sk, and then re-encrypting sk with PK_2 does not have such efficient verifiability. We could try to attach a non-interactive zero-knowledge (NIZK) proof for this purpose, but such proof might be quite inefficient, especially with chosen *ciphertext* secure encryptions c_1 and c_2.

Encapsulated VRF. We replace the VRF in the above approach with a new primitive known as an encapsulated verifiable random function. Intuitively, an EVRF allows the phone to publish a public key PK. The desktop can generate a ciphertext C and a trapdoor T, from just PK such that the EVRF value can be computed by the desktop using PK and T, while the phone can use SK and C to generate the same value. Now, the desktop erases T after the indexing process. Further, the phone that produces y to authorize the search also produces proof z to show that y is correct.

1.3.2 Our Results

Our contributions in this domain are as follows:

- We introduce the new primitive known as encapsulated search index (ESI) in Section 8.2.1. We also present extensions of this primitive that allows for the distribution of trust in Section 8.2.3, the delegation of search in Section 8.2.4, and an update of the index in Section 8.2.5.

- We introduce a new primitive of independent interest known as encapsulated verifiable random function in Section 8.3.1. We also present natural extensions of this primitive to allow for non-interactive threshold construction in Section 8.4.1 and delegation in Section 8.5.1.

- We present a generic construction of ESI from any EVRF and a non-private search dictionary in Section 8.3.2 and present security proofs for the same. Further, our generic result

inherits the properties of threshold and distribution from the EVRF, helping us achieve the desired ESI extensions.

- We present our construction of the standard EVRF in Section 8.3.4 and of the threshold EVRF in Section 8.4.2. We also present various flavors of delegatable EVRF in Section 8.5.2, Section 8.5.3, and Section 8.5.4.

- We prove the security of our EVRFs under a variety of assumptions.

1.3.3 Related Work

If one were to look at the problem of searchable encryption through the roles that are being played. We have three roles - index creator, search approver, and storage location. We will focus on the first two, for this case.

Searchable Symmetric Encryption. *Searchable Symmetric Encryption* (SSE) [DWP00, Goh03, CM05, CGKO06, DWP00, BC04] was introduced to help solve the problem of searchable encryption in the symmetric key setting. In this case, we have that the index creator and the search approver share the same key (or are the same person). In other words, we have a single reader and a single writer of the index. Of importance is that the sharing of the key helps us unlock the power of a non-private dictionary and achieve sublinear search time. Additionally, they allow for "universal indexing" (or "universal searching") where a single token z_w for keyword w is the same for every document (i.e., there is no document-specific handle that is required). Note that any "universal indexing" scheme can be made a document-specific indexing scheme by prefixing the document ID for the indexing of document D.

Another important property is the so-called "hidden queries" [DWP00] the idea of "keyword-privacy". Formally, the requirement is for an adversary who, given an index E and keyword token

z is unable to distinguish if z is for keyword w_0 or w_1.[7] Importantly, this property is incompatible with universal indexing, in the public-key setting. An adversary can always generate the index (which requires only the public key, as discussed below) for some document D_0 containing w_0 and not w_1 and then test if z works on this index.

Fortunately, for our motivating application, neither of these two properties is required. The document is available at the time of indexing and therefore a document-specific keyword suffices and the verifiability property implies that we cannot have keyword privacy. Further, if Alice was to use SSE to solve the problem, she would have to transmit the entire document to the phone and then have the phone index and send the index back to the desktop. This is inefficient. The other option would be for Alice to have the secret key shared with the desktop as well, but then more trust is needed for the Desktop and also requires a new secret key for every document. In other words, we see that Alice desires to generate the entire encrypted index E on her desktop without communicating with the phone. The only communication needs to happen with the phone to authorize searches which naturally implies a public-key cryptography-based solution.

Public Key Encryption with Keyword Search. Public Key Encryption with Keyword Search (PEKS) [BDOP04, ABC+05, BSNS08, BKOS07, RPSL09, ZLT+20] can be viewed as the public-key counterpart of SSE. Alice publishes a public-key PK which allows anybody to create an encrypted index for her. In other words, we have several index-creators (i.e., multiple writers) but a single search approver (i.e., reader). PEKS also requires universal indexing and as remarked earlier cannot achieve keyword privacy. More importantly, as an artifact of universal indexing, the system is inherently slow. The only way to test would be to test each "ciphertext" one by one on whether it matches the keyword being searched. Fortunately, because indexing can be document-specific ESI helps us achieve sublinear search time.

Summarizing the above discussion (see Table 1.1), we can highlight five key properties of a

[7] It is important to note that *all* SSE schemes in the literature do not achieve the strongest possible keyword privacy and suffer from various forms of information leakage [CGPR15, CGKO06, KPR12].

given searchable encryption scheme: public-key indexing, sublinear search, universal indexing, keyword privacy, and index privacy. All of ESI/SSE/PEKS satisfy (appropriate form) of index privacy and differ — sometimes by choice (ESI) or necessity (PEKS) — in terms of keyword privacy. So the most interesting three dimensions separating them are public-key indexing, sublinear search, and universal search, where (roughly) each primitive achieve two out of three. For our purposes, however, *ESI is the first primitive which combines public-key indexing and sublinear search*, which is precisely the setting of our motivating example. This forms the backbone of the commercially deployed product called Atakama[Ata22].

	SSE	PEKS	ESI
Public-Key Indexing	✗	✓	✓
Sublinear Search	✓	✗	✓
Universal Index	✓	✓	✗
Index Privacy	✓	✓	✓
Keyword Privacy	✓ (partial)	✗ (impossible)	✗ (by choice!)

Table 1.1: A comparison of SSE, PEKS, and ESI.

1.4 Updatable Public Key Encryption

For purposes of privacy, forward security refers to the property by which a system continually updates secret information to ensure that a compromise of the secret information at a period t does not render any communication in prior periods insecure. This concept was initially proposed in the context of signatures [] but has grown in its relevance in the context of encryption schemes. In either regime, one takes time and divides it into smaller periods or epochs with the secret information continually updated from one period to the next.

Forward Security is easy to achieve in the symmetric-key world, provided the sender and the receiver can stay synchronized [BY03]. This solution requires the use of a pseudorandom generator (PRG) G, where given the current state s, the sender produces $(r, s') \leftarrow G(s)$ to get the one-time symmetric key r and the new state s'. Then, compromise of s' does not render r in-

secure. However, the solution is not that straightforward in the public key encryption scheme.

Forward-Secure PKE. The initial paper of Canetti *et al.*[CHK03], which is still essentially state-of-the-art, defined FS-PKE as follows: from an initialized $(\mathsf{pk}_0, \mathsf{sk}_0)$ public key-secret key pair, one can define two synchronized chains $\mathsf{pk}_0 \rightarrow \mathsf{pk}_1 \rightarrow \mathsf{pk}_2 \rightarrow \ldots$ and $\mathsf{sk}_0 \rightarrow \mathsf{sk}_1 \rightarrow \mathsf{sk}_2 \rightarrow \ldots$ with the property that they can be *independently* produced by multiple senders and a single receiver respectively. The consistency requires that any message encrypted by pk_i is decrypted correctly by sk_i. Finally, the compromise of sk_j does not render any message encrypted by pk_i for $i < j$, which captures the forward-security requirement. Their work also showed how to build FS-PKE from the Hierarchical Identity-Based Encryption (HIBE) [HL02, GS02] scheme. This generic construction implied that with further research on actual constructions of HIBE Schemes [BB04, CHKP12, BBG05, DG17b, DG17a, BLSV18], we also obtained FS-PKE constructions under various assumptions. These constructions also included theoretical schemes from fundamental assumptions, like DDH/CDH, factoring, and super-low-noise LPN [DG17b, DG17a, BLSV18]. Given the wide assortment of FS-PKE constructions, it is important to stress that these are inherently complicated and inefficient compared to actual PKE constructions from the same assumptions. Closing this efficiency gap is an open problem.

Updatable PKE. In this work, we look at another approach to achieving forward security in the public key encryption setting. The primitive, known as Updatable PKE (UPKE), can be viewed as an attempt at closing the efficiency gap, albeit under a relaxation of FS-PKE. UPKE was proposed by Jost *et al.*[JMM19], motivated by independent applications to secure group messaging. It was later used by Alwen *et al.*[ACDT20] for their construction of a secure group messaging protocol. The idea here is that rather than expecting the two synchronized chains to be generated independently, a sender initiates a "key update" by sending an "update ciphertext" and updating the receiver's public key. Upon processing the update ciphertext, the receiver updates its secret key with the expectation that the new secret key is consistent with the updated public key. While

we expect an honest sender to generate good randomness for the update ciphertext, a malicious sender cannot harm the receiver with bad randomness. Further, an honest sender is guaranteed that once the receiver updates its secret key using the update ciphertext, compromising the resulting secret key would not render prior communications insecure. It is important to note that the setting of UPKE assumes an ordered sequence of these update ciphertexts, which the receiver will then process, and this sequence is available to all senders. This trivially holds for the two-party setting where all communications are sent by the same sender [JMM19] (similar to the symmetric-key setting), or we can assume some external serialization mechanism for multiple sender settings, as seen in the work by Alwen *et al.* [ACDT20].

Informal Syntax and Security of UPKE. Let us now be more precise. In addition to the standard algorithms of a PKE - key generation, encryption, and decryption - we have two additional algorithms Upd-Pk and Upd-Sk.[8] A sender runs Upd-Pk on the current public-key pk_{i-1} to produce an *update ciphertext* up_i and a new public key pk_i. Upon receiving this communication, the receiver runs Upd-Sk on the current secret key sk_{i-1} and update ciphertext up_i, to produce the new secret-key sk_i.

The security requirement is that exposure of any key sk_i should not harm the security of messages encrypted under public keys pk_j, for any $j < i$, *provided at least one "good" update happened from period j to i*. Here, by "good update" we mean that the attacker did not compromise the randomness used by the sender to generate this update. In the earlier works [JMM19, ACDT20], as an artifact of secure messaging application assumptions, they assumed that the senders were honest, but the attacker could compromise the randomness used by the sender. However, we look at a stronger security assumption where we allow for the adversary to provide malicious randomness for the "bad updates". To model forward security, our security game allows for the adversary to provide randomness $\delta_1, \ldots, \delta_q$ with the challenger executing the update procedures consistent

[8] Note that in our discussions we use a different syntax from the earlier works [JMM19, ACDT20], but it is easy to re-cast our syntax in the context of their syntax.

with $\delta_1, \ldots, \delta_q$ to obtain public key chain $\mathsf{pk}_1, \ldots, \mathsf{pk}_q$ and secret key chain $\mathsf{sk}_1, \ldots, \mathsf{sk}_q$. pk_q is used to encrypt the challenge message m_b. Further, the challenger samples a fresh randomness δ^* and runs the update procedures concerning this delta to get the new public key $\mathsf{pk}^*, \mathsf{up}^*, \mathsf{sk}^*$. The adversary is now provided with these values and the challenge ciphertext.

Prior Construction of UPKE. Recall that we motivated UPKE to achieve a weaker definition of FS-PKE while obtaining better efficiency. Indeed, the works of [JMM19, ACDT20] presented a simple and fast (orders of magnitude faster than HIBE-baed FS-PKE schemes) construction based on the CDH assumption in the random oracle model (ROM). At its core, they use the standard hashed ElGamal encryption scheme. Given a public key $h = g^s$, we encrypt a message m as $(c = g^r, w = Hash(h^r) \oplus m)$. Now, with secret key s, one can decrypt as (c, w) outputs $w \oplus Hash(c^s)$. For the update procedures, the sender chooses a random exponent δ, and simply *encrypts δ using the encryption as mentioned above procedure.* Leveraging the homomorphic properties, one can set the new public key as $h' = h \cdot g^{\delta}$, implicitly setting the new secret key as $s' = s + \delta$. The receiver can decrypt the update ciphertext to compute δ and update the secret key on its end.

If we looked at the security of the hashed ElGamal-based construction [JMM19, ACDT20], the adversary receives as input $\mathsf{sk}^* = s' = s + \delta$ (i.e., along with encryption of δ as up^*). Therefore, the adversary receives both an encryption of a function of the secret key ($\delta = s' - s$) and a leakage of the secret key ($s' = s + \delta$). Fortunately, for a random δ, s' is trivial leakage, while the random oracle easily handled (so-called) key-dependent-message (KDM) security [BRS03]. Therefore, the random oracle model is critically used to break the circularity.

This work looks at how to build constructions in the standard model.

1.4.1 Our Approach

We need a construction that handles the leakage of the secret key and the security of the message dependent on the key. In particular, we will rely on circular-secure encryption schemes

in the standard model under DDH/LWE (e.g., [BHHO08, ACPS09]) where the key s and/or the updates δ must consist of "small" values in some larger group $\mathbb{Z}_p$ and hence the leakage $s' = s + \delta$. However, the leakage will no longer be trivial.[9] Luckily, these schemes are also leakage resilient; hence, this non-trivial leakage does not hurt security. Indeed, our constructions will follow the following template that builds on a regular PKE that has the following properties:

1. *Circular-secure and leakage-resilient (CS+LR):* Given encryption of the secret key s and any bounded-entropy leakage on s, the scheme is still semantically secure. See Section 9.3 for the precise definition.

2. *Key-Homomorphic*: Let (c, pk) be a public key and a ciphertext pair that corresponds to some secret key sk. Then, given some offset δ, we can convert them into a public key and a ciphertext pair (pk', c') that corresponds to the secret key $\mathsf{sk}' = \mathsf{sk} + \delta$, while preserving the encrypted message. In other words, we have the following algorithm: $(\mathsf{pk}', c') \leftarrow KH(\mathsf{pk}, c; \delta)$.

3. *Message-Homomorphic*: Let e be ciphertext encrypting some value s. Then, given some offset s', we can convert it into a ciphertext e' encrypting $s' - s$. In other words, we have the following algorithm: $e' \leftarrow MH(e; s')$.

Now, we briefly outline our reduction idea though the actual details vary between our constructions. Let us assume that we have a UPKE attacker $\mathcal{A}$, and we will build a CS+LR attacker $\mathcal{A}'$. Our reduction idea will be for the case where $q = 1$, i.e., the adversary provides single randomness for the update. This is presented in Figure 1.1.

[9] This is true for the DDH-based scheme of Boneh *et al.*[BHHO08] since circular security requires encrypting in the exponent and decryption involves solving discrete log; therefore the encrypted values must be small. This is also true for the LWE-based scheme, where the secret key must be small for correctness.

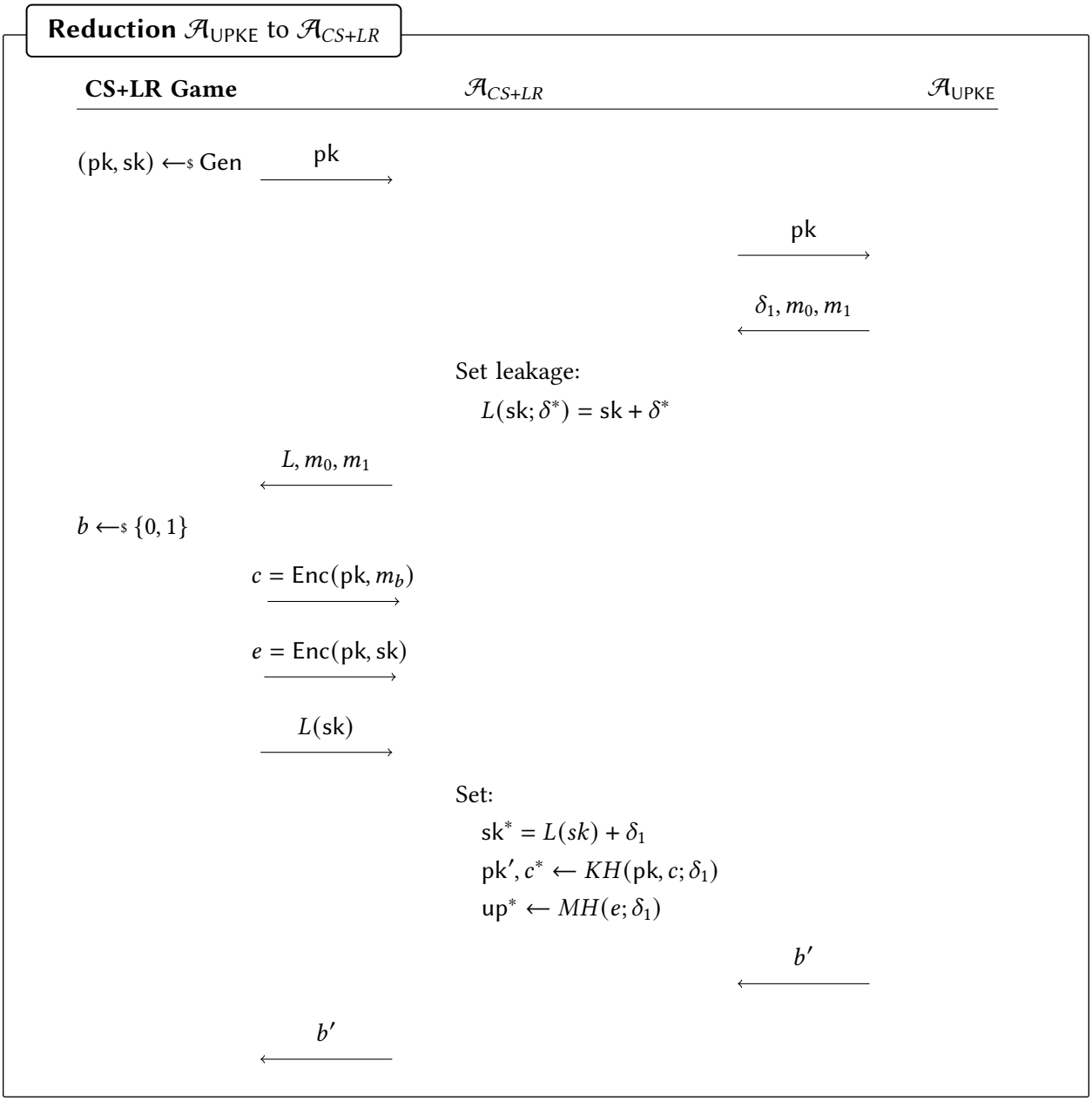

Figure 1.1: A reduction idea for building an attacker $\mathcal{A}_{CS+LR}$ against the CS+LR security game for the underlying PKE, using an attacker $\mathcal{A}_{UPKE}$ that breaks the UPKE security. Here, we set the leakage function as a probabilistic leakage function $\mathsf{sk}+\delta^*$ where δ^* is chosen by the challenger and is unknown to $\mathcal{A}_{CS+LR}$. This leakage offset is implicitly set as the final update's δ^*.

1.4.2 Our Results

Our main contribution is presenting two efficient UPKE schemes in the standard model from the DDH and the LWE Assumption.

- First, we show that the BHHO cryptosystem [BHHO08] constructed from the DDH assumption satisfies the properties we need. In their construction, the secret key sk is a vector $\boldsymbol{s} \in \mathbb{Z}_p^\ell$. It achieves circular security holds as each component of the secret key is encrypted in the exponent with decryption recovering the secret key by merely the discrete log. Therefore, it requires that a "short" $\boldsymbol{s} \in \{0,1\}^\ell \subseteq \mathbb{Z}_p^\ell$. They also need their message space to also be a bit string. In our construction, we initialize $\boldsymbol{s}_0 \leftarrow\!\!\$\ \{0,1\}^\ell$. For the update, we choose offset $\boldsymbol{\delta}_i \leftarrow\!\!\$\ \{0,1\}^\ell$ and is encrypted bit-by-bit in the exponent; the updated secret key will be $\boldsymbol{s}_{i+1} = \boldsymbol{s}_i + \boldsymbol{\delta}_i$ where the addition is performed over $\mathbb{Z}_p$. The construction can be found in Section 9.4. Note that his construction was shown to be circular secure [BHHO08] and leakage-resilient [NS09]; we show that the two security properties also hold simultaneously in Section 9.4.2. When we use this scheme as a UPKE, we rely on the fact that for randomly chosen $\boldsymbol{\delta}, \boldsymbol{s} \in \{0,1\}^\ell$, the leakage value of $\boldsymbol{\delta} + \boldsymbol{s}$ (with addition over $\mathbb{Z}_p$) only reduces the entropy of $\boldsymbol{s}$ by $\ell \cdot \log(4/3) \leq \ell/2$ bits. We prove this as Claim 9.18. Finally, in Section 9.4.4, we show that the construction is indeed a secure UPKE.

- Second, we show that the dual-Regev cryptosystem [Reg05, GPV08] constructed from the LWE assumption also satisfies the properties we need. In their construction, the secret key sk is a vector $\boldsymbol{s} \in \mathbb{Z}_p^\ell$. The construction can be found in Section 9.5. We show that the construction is CS+LR security, similar to the DDH results, in Section 9.5.2. When we use this scheme as a UPKE, we encounter the following issue: while the scheme is key-homomorphic, when we update the key we obtain an incorrect ciphertext distribution – in particular, the "error term" distribution is perturbed. This necessitates deploying the "noise smudging" technique (see Lemma 9.2) where we add super-polynomial noise to the

ciphertext to hide the smaller difference in the error term, which is polynomial.

Our resulting constructions are significantly more efficient and less complicated when compared to the HIBE schemes from *the same assumptions*. A rough summary of efficiency, security, usability, and assumptions trade-off for our schemes when compared to previous PKE, UPKE, and FS-PKE schemes is given in Table 1.2. It is clear from the table that our efficiency falls between that of PKE and FS-PKE (much closer to the former), we achieve the same (resp. much stronger) forward security as FS-PKE (resp. PKE), but we do require a stronger synchronization assumption than FS-PKE.

Factors	**PKE**	**UPKE** [JMM19, ACDT20] RO Model	**UPKE** (this work) Standard Model	**FS-PKE**
Efficiency	Very Efficient	$\approx$ PKE	$\approx$ PKE $\cdot \kappa$	Inefficient[10] (from HIBE)
Assumptions	DDH/CDH, Factoring, LWE	CDH	DDH, LWE	DDH/CDH, Factoring, LWE
Forward Security?	No	Yes	Yes	Yes
Synchronization	None	Strong (Updates)	Strong (Updates)	Weak (Time Periods)

Table 1.2: Comparison of Different Primitives. (κ is the security parameter.)

1.4.3 Related Work

Hierarchical Identity-Based Encryption (HIBE). Canetti *et al.* [CHK04] showed how to build FS-PKE (and therefore also UPKE) from any Hierarchical Identity-Based Encryption (HIBE) [HL02, GS02, BB04, CHKP12, BBG05, DG17b, DG17a, BLSV18].

The HIBE construction from DDH/CDH [DG17b, DG17a, BLSV18] yields an alternate construction of UPKE from DDH/CDH in the standard model. However, this construction is mainly of theoretical interest and is hugely impractical. Their construction critically relies on complex garbled circuits that perform public-key operations. In more detail, their construction relies on a chain of $O(\kappa)$ garbled circuits for security parameter κ. Now, each of these circuits outputs $O(\kappa)$

special ciphertexts (encrypted labels for next level garbled circuit) with each ciphertext containing at least $O(\kappa)$ group elements. Since this computation is inside a garbled circuit, we are left with at least another $O(\kappa)$ overhead on top. This does not account for another $O(\kappa)$ overhead to go from HIBE to FS-PKE/UPKE, yielding a total complexity of at least $O(\kappa^5)$. In other words, this resulting construction is at least $O(\kappa^3)$ worse than our scheme, ignoring concrete overheads.

On the other hand, building HIBE from LWE [CHKP12, ABB10] yields an alternate construction of UPKE from LWE in the standard model. The resulting schemes could potentially be piratically efficient. Yet, our construction is still significantly simpler and more efficient for several reasons: (1) It does not rely on lattice trapdoors or GPV style pre-image sampling [GPV08], making our scheme conceptually simpler and practically more efficient. (2) Our secret key is a single lattice vector rather than an entire lattice basis. This shortens our secret keys by roughly an $O(\kappa)$ factor. (3) We avoid the additional $O(\kappa)$ factor overhead in the transformation from HIBE to FS-PKE/UPKE.

Forward-secure Signatures. Forward-Secure Signatures [And97], much like FS-PKE, requires that any compromise of the current signing key does not enable forgery of messages for previous periods. In particular, the tree-based FS-signature scheme of Bellare and Miner [BM99] (later extended by Malkin et al. [MMM02]) inspired the work of HIBE-based FS-PKE of [CHK04]. There are also constructions in the random oracle setting [AR00, IR01, KR03].

Related Key Evolving Encryption Schemes. The works of Jaeger and Stepanovs [JS18] and Poettering and Rössler [PR18] proposed two related notions of *key-updatable* PKE scheme. These constructions provide an even stronger form of key evolution than FS-PKE where key updates can be labeled by arbitrary, possibly adversarially chosen, strings. These schemes were also built from HIBE.

Circular and KDM Secure Encryption Schemes. Circular-secure schemes allow the attacker to see encryptions of the scheme's secret key. A natural extension of this notion studies a cycle of $(\mathsf{sk}_i, \mathsf{pk}_i)$ pairs for $i = 1, \ldots, n$ where we encrypt sk_i under $\mathsf{pk}_{i \bmod n+1}$. This was defined as *key-dependent message security (KDM)* by Black et al. [BRS03] and as *circular security* by Camenisch and Lysyanskaya [CL01]. The first cryptosystem in the standard model with proof of KDM-security under a standard assumption was given by Boneh et al. [BHHO08]. Subsequently, constructions from the learning with errors (LWE) [ACPS09] and quadratic residuosity [BG10] assumptions were proposed. Construction for identity-based KDM-secure encryption [AP12] was also proposed. While the construction of Boneh et al. [BHHO08] was for affine functions, subsequent "KDM amplifications" transforms extended the class of functions significantly [BHHI10, BGK11, MTY11, App14].

Leakage-Resilient Encryption Schemes. Most of the security models do not capture possible *side-channel attacks*. These attacks are designed to exploit unintended leakage that often stems from the physical environment. Akavia et al. [AGV09] proposed a realistic framework that aimed to capture information about the leakage. Subsequent work by Naor and Segev [NS09] analyzed the resilience of public-key cryptosystems to leakage. An important result was that they showed the (even slightly optimized version of the) BHHO scheme [BHHO08] was resilient to $|\mathsf{sk}|(1 - o(1))$ bits of leakage. Subsequent work [DGK+10] showed the leakage resilience of both the BHHO scheme and the dual Regev encryption scheme [Reg05, GPV08] in the auxiliary input model. Brakerski et al. [BLSV18] studied both the leakage resilience and circular security of anonymous IBE. We point to the survey of leakage resilient cryptography by Kalai and Reyzin [KR19] for additional work in this domain.

Different "Updatable" Encryption. With an unfortunate naming collision, there has been a different kind of "updatable encryption schemes" considered in the literature [BLMR13, EPRS17, LT18, KLR19, BDGJ20, BEKS20]. These are *symmetric-key* encryption schemes that aim to ac-

complish key rotation in the cloud, specifically moving the ciphertexts under the old key to the new key. In particular, these schemes produce multiple encryptions of the same message under different keys and aim to produce update tokens that allow the update of old ciphertexts, without leaking the message content. In contrast, updatable schemes in this paper are *public-key*, encrypt different messages, and aim to achieve forward security. Thus, the notions are very different despite the partial naming collision.

1.5 Outline and Motivation of this Work

Our Motivation. As mentioned earlier, this work is an attempt at providing theoretical approaches and solutions to problems that are inspired by practical applications. To put it succinctly, the crux of this work is the theory applied to practice-oriented problems. This work is spread over four parts where each deals with a particular problem that is inspired or of practical interest. Part I revisits the problem of random number generation. There is a rich literature of theoretical work in this domain and an equal quantity of practical constructions of PRNGs. However, there is a gap. The theoretical constructions are not deployed in practice, and theoretical proofs of security for the real-world constructions are lacking on various axes. We aim to bridge this gap through this work. Part II focuses on the problem of understanding the security of a key symmetric-key component of block ciphers, specifically AES. There is a large body of literature that has taken an assortment of approaches to prove the security of such block ciphers. However, these analyses have either been too coarse to capture actual constructions, or abstracted away key components that have resulted in "proofs of security" of insecure constructions. This work introduces a new framework that provides concrete (and plausible) security estimates for block ciphers. Part IV, perhaps, provides the most tangibly practical result where the protocol analyzed is a commercially available product. We motivate a new slant to the problem of searchable encryption by introducing new primitives to solve this problem. Finally, in Part III we look at the

problem of forward-secure encryption and its simpler case of updatable public key encryption (UPKE). UPKE, as a primitive, plays a key role in the practical problem of secure group messaging, as proposed by Alwen *et al.* [ACDT20]. The focus of this work is to look at solutions that are (a) secure against quantum adversaries, and (b) do not rely on the random oracle model. While our results in this setting might not be available for immediate deployment owing to large parameter sizes, we do present a recipe or a framework to build UPKE from standard public-key encryption that satisfies some properties.

Outline. In Chapter 2, we present some preliminaries. In Chapter 3, we discuss seedless extraction. In Chapter 4, we introduce the primitive known as seedless pseudorandom number generators with input and formally define the security of this primitive. We also define intermediate security definitions and prove how these different intermediate definitions compose for the overall security. In Chapter 5, we present our constructions of seedless PRNGs and prove their security, in various idealized models of computation. Finally, in Chapter 6, we look at the problem of achieving security against premature next attacks when using seedless PRNGs. Our results include both an impossibility result and various positive results that have a direct impact on our understanding of actual PRNGs that form a part of various operating systems. In Chapter 7, we introduce a new framework that can be used to provide concrete hardness estimates for block ciphers. In Chapter 9, we introduce a new primitive that helps achieve sublinear search time, in the public-key setting for the problem of searchable encryption. We also present other primitives and extensions in this chapter. Finally, in Chapter 6, we revisit the primitive known as updatable public key encryptions, presenting candidate constructions in the standard model.

2 Preliminaries

2.1 Notation

Let PPT denote probabilistic polynomial time. We denote by κ the security parameter. We denote by $\mathrm{negl}(\kappa)$ a function that is negligible in the security parameter κ. Additionally, when we say that a value Z is "negligible", it implies that $Z < \mathrm{negl}(\kappa)$.

Definition 2.1 (Negligible Function)**.** A function negl is negligible iff $\forall c \in \mathbb{N}$, $\exists n_0 \in \mathbb{N}$ such that $\forall n \geq n_0$, $\mathrm{negl}(n) < n^{-c}$.

For a distribution X, we use $x \leftarrow\!\!\$\, X$ to denote that x is a random sample drawn from the distribution X. For a set S we use $x \leftarrow\!\!\$\, S$ to denote that x is chosen uniformly at random from the set S. We denote by U_d the uniform distribution over $\{0,1\}^d$.

2.1.1 Information-Theoretic Notations

- The *statistical distance* of two random variables X and Y is

$$\mathrm{SD}(X,Y) = \frac{1}{2}\sum_{x} \left|\Pr[X=x] - \Pr[Y=y]\right|.$$

- The *prediction probability* of a random variable X is $\mathrm{Pred}(X) := \max_x \Pr[X=x]$, and we also denote $\mathrm{Pred}(X|y) := \max_x \Pr[X=x|Y=y]$.

- The conditional version of prediction probability is defined as

$$\mathsf{Pred}(X|Y) \;:=\; \mathbf{E}_{y\leftarrow Y}\left[\mathsf{Pred}(X|y)\right] .$$

- The *(average-case) conditional min-entropy* is

$$\mathrm{H}_\infty(X|Y) = -\log(\mathsf{Pred}(X|Y)).$$

2.1.2 Security Games

All of the security properties considered in this paper are captured by considering a game between a challenger and an attacker $\mathcal{A}$, both of which may have access to an ideal primitive P. The goal of the attacker is to guess a random bit b chosen by the challenger, who offers a set of oracles to the attacker to aid with this task. The *advantage* of $\mathcal{A}$ is defined as

$$2 \cdot \left| \Pr[\mathcal{A} \text{ wins}] - 1/2 \right|,$$

where the probability is over the randomness of $\mathcal{A}$, of the challenger, and of the ideal primitive. The cases where $b = 0$ and $b = 1$ are referred to as the *real world* and the *ideal world*, respectively. One may equivalently consider $\mathcal{A}$'s advantage at telling these two worlds apart, i.e.,

$$\left| \Pr[\mathcal{A} = 1|b = 0] - \Pr[\mathcal{A} = 1|b = 1] \right| .$$

Part I

Random Number Generation, Revisited

3 Seedless Extraction and Key Derivation

This chapter is based on joint work with Sandro Coretti, Yevgeniy Dodis, and Stefano Tessaro that appeared in CRYPTO 2019 [CDKT19a]. Some passages are taken verbatim from the full version of this paper [CDKT19b].

We begin by looking at the problem of *extraction*, i.e., producing uniformly random bits from weak high-entropy sources. Extraction can be seen as corresponding to the post-compromise security of PRNGs, and as such it will be implied by PRNG *robustness* (as defined in Section 4.1.2). The definition of extraction security in Section 3.2 considers the entropy of the attacker's input to the extractor conditioned on the attackers *state* and the *queries* made to an ideal primitive P. A definition is provided for *computational* or *information-theoretic* security. IT extractors differ from computational ones in that the output of the extractor remains random even if the attacker, *after providing the input*, is given the entire function table of the underlying ideal primitive. That is, IT extractors achieve so-called *everlasting security* (cf. works in the hybrid bounded-storage model by Harnik and Naor [HN06]).

Section 3.3 considers extracting with a *monolithic random oracle*. The corresponding security proofs (for the computational and IT cases) are instructive for understanding the actual PRNG constructions provided in Section 5. Since considering a monolithic oracle is not motivated by any hash function used in practice, Section 3.4 introduces the concept of *online* extraction. An

online extractor accumulates the entropy of its inputs in an internal state, from which uniform randomness can be produced. Finally, in order to illustrate the non-triviality of online extraction, Section 3.5.5 shows that extractors based on the popular CBC mode are not suitable for extraction.

Due to their importance, Section 3.5.6 briefly discusses *key-derivation functions (KDFs)* and analyzes the Extract-then-PRF paradigm (on which the widely used HKDF is based) put forth by Krawczyk [Kra10] in the seedless setting.

3.1 Preliminaries

3.1.1 The H-Coefficient Technique

When considering an adaptive distinguisher $\mathcal{D}$ that tries to tell apart two different worlds, usually termed *real* and *ideal* experiments, the H-coefficient technique [Pat09] is a handy tool for analyzing the distinguishing advantage of $\mathcal{D}$.

The H-coefficient technique considers transcripts between distinguisher $\mathcal{D}$ and the challenger. These transcripts are partitioned into two groups: the *good* transcripts and the *bad* transcripts. For good transcripts τ, an H-coefficient proof will commonly derive a lower bound on the ratio of the probability of τ occurring in the real world and that of τ occurring in the ideal world. For bad transcripts, which are normally defined as the transcripts for which said lower bound cannot be derived, one upper bounds the probability that they occur. This latter bound can be proved in the *ideal* world, which usually greatly simplifies the derivation.

Transcripts. The interaction of $\mathcal{D}$ with either the real or the ideal experiment produces a transcript T that contains the queries made by $\mathcal{D}$ and the corresponding answers given by the challenger. For a fixed transcript τ, denote by $p_0(\tau)$ and $p_1(\tau)$ the probabilities that the real and ideal experiments, respectively, the challenger produces the answers in τ if asked the queries in

τ.[1] Similarly, the behavior of a distinguisher $\mathcal{D}$ is described by a function $p_{\mathcal{D}}(\tau)$ that assigns to τ the probability that $\mathcal{D}$ produces the queries in τ if given the answers in τ. Observe that, therefore, the probability of a particular transcript τ occurring in an interaction of $\mathcal{D}$ with the real experiment is $\Pr[T_0 = \tau] = p_{\mathcal{D}}(\tau) \cdot p_0(\tau)$ and, similarly, $\Pr[T_1 = \tau] = p_{\mathcal{D}}(\tau) \cdot p_1(\tau)$ for the ideal world. In the following, denote by $\mathcal{T}$ the set of all transcripts τ.

Bounding the distinguishing advantage. The distinguishing advantage of $\mathcal{D}$ is upperbounded by the statistical distance

$$\begin{aligned}
\mathrm{SD}(T_0, T_1) &= \sum_{\tau \in \mathcal{T}} \max\{0, \Pr[T_1 = \tau] - \Pr[T_0 = \tau]\} \\
&= \sum_{\tau \in \mathcal{T}} \max\{0, p_{\mathcal{D}}(\tau) \cdot p_1(\tau) - p_{\mathcal{D}}(\tau) \cdot p_0(\tau)\} \\
&= \sum_{\tau \in \mathcal{T}} p_{\mathcal{D}}(\tau) \cdot p_1(\tau) \left(1 - \frac{p_0(\tau)}{p_1(\tau)}\right) \\
&= \sum_{\tau \in \mathcal{T}} \Pr[T_1 = \tau] \left(1 - \frac{p_0(\tau)}{p_1(\tau)}\right) . \qquad (3.1)
\end{aligned}$$

Suppose that for some set $\Gamma \subseteq \mathcal{T}$ of *good* transcripts, a lower bound

$$\frac{p_0(\tau)}{p_1(\tau)} \geq 1 - \varepsilon$$

is known for all $\tau \in \Gamma$ and some $\varepsilon \geq 0$. Then, (3.1) becomes

$$\begin{aligned}
\sum_{\tau \in \mathcal{T}} \Pr[T_1 = \tau] \left(1 - \frac{p_0(\tau)}{p_1(\tau)}\right) &\leq \sum_{\tau \in \Gamma} \left(1 - \frac{p_0(\tau)}{p_1(\tau)}\right) + \sum_{\tau \in \mathcal{T} \setminus \Gamma} \Pr[T_1 = \tau] \\
&\leq \varepsilon + \Pr[T_1 \in \mathcal{T} \setminus \Gamma] ,
\end{aligned}$$

where transcripts $\tau \in \mathcal{T} \setminus \Gamma$ are commonly referred to as *bad* transcripts. Given the above, apply-

[1] Observe that $p_0(\tau)$ and $p_1(\tau)$ depend only on the corresponding experiment and are independent of $\mathcal{D}$.

ing the H-coefficient technique entails defining a set of good transcripts, bounding the fraction above, and showing that bad transcripts are unlikely. Note that the latter can be done in the ideal experiment, which is usually considerably easier than doing so in the real experiment.

Theorem 3.1 (H-coefficient method)**.** *For two experiments described by* $\mathrm{p}_0(\cdot)$ *and* $\mathrm{p}_1(\cdot)$*, respectively, if there exists a set* $\Gamma \subseteq \mathcal{T}$ *and* $\varepsilon, \delta \geq 0$ *satisfying*

1. ***(ratio analysis)*** $\mathrm{p}_0(\tau)/\mathrm{p}_1(\tau) \geq 1 - \varepsilon$ *for all* $\tau \in \Gamma$ *and*
2. ***(bad event analysis)*** $\Pr[T_1 \notin \Gamma] \leq \delta$*,*

then the distinguishing advantage of any distinguisher $\mathcal{D}$ *is bounded by* $\varepsilon + \delta$*.*

3.1.2 Information-Theoretic Preliminaries

The *collision probability* of a random variable X is defined simply as $\mathsf{Coll}(X) := \sum_x \Pr[X = x]^2$. Moreover, let $\mathsf{Coll}(X|y) := \sum_x \Pr[X = x|Y = y]^2$, and define the conditional collision probability

$$\mathsf{Coll}(X|Y) \;:=\; \mathbf{E}_{y \leftarrow Y}\big[\mathsf{Coll}(X|y)\big]\;.$$

The following two propositions relate the statistical distance from uniform to the collision probability. The first is well known and at the core of the proof of the leftover hash lemma [ILL89]; the latter is proved for self-containment.

Proposition 3.2. *For any random variable* X *with size-*N *range, and a uniformly distributed* U *with the same range,*

$$\mathrm{SD}\,(X, U) \;\leq\; \frac{1}{2}\sqrt{N \cdot \mathsf{Coll}(X) - 1}\;.$$

Proposition 3.3. *Let* F *be chosen uniformly at random from a set* $\mathcal{F}$*. Then, for any random variable* X *with size-*N *range (arbitrarily correlated with* $\mathcal{F}$*), and a uniformly distributed* U *with the same*

range, independent of F,

$$\mathrm{SD}\left((X,F),(U,F)\right) \;\leq\; \frac{1}{2}\sqrt{N\cdot \mathrm{Coll}(X|F)-1}\,.$$

Proof. By Proposition 3.2,

$$\mathrm{SD}\left((X,F),(U,F)\right) \;\leq\; \frac{1}{2}\sqrt{N|\mathcal{F}|\cdot \mathrm{Coll}(X,F)-1}\,.$$

Moreover,

$$\begin{aligned}\mathrm{Coll}(X,F) &= \sum_{x,f}\Pr[(X,F)=(y,f)]^2\\ &= \frac{1}{|\mathcal{F}|}\sum_{f}\Pr[F=f]\sum_{x}\Pr[X=x|F=f]^2\\ &= \frac{\mathrm{Coll}(X|F)}{|\mathcal{F}|}\,,\end{aligned}$$

from which the proposition follows. □

The following proposition will also be useful.

Proposition 3.4. *Consider two random variables X and Y with identical range and let $\mathcal{E}$ and $\mathcal{E}'$ be events on their respective probability spaces. Assume $\Pr[\mathcal{E}]=\Pr[\mathcal{E}']$, then*

$$\mathrm{SD}\left(X,Y\right) \;\leq\; \mathrm{SD}\left(X|\mathcal{E},Y|\mathcal{E}'\right)+\Pr[\overline{\mathcal{E}}]\,.$$

Proof. Observe that

$$\begin{aligned}
\mathrm{SD}\,(X,Y) &= \frac{1}{2}\sum_x |P_X(x) - P_Y(y)| \\
&= \frac{1}{2}\sum_x \left| P_{X|\mathcal{E}}(x)\mathrm{Pr}[\mathcal{E}] + P_{X|\overline{\mathcal{E}}}(x)\mathrm{Pr}[\overline{\mathcal{E}}] - P_{Y|\mathcal{E}'}(x)\mathrm{Pr}[\mathcal{E}'] - P_{Y|\overline{\mathcal{E}'}}(x)\mathrm{Pr}[\overline{\mathcal{E}'}] \right| \\
&\leq \mathrm{Pr}[\mathcal{E}] \cdot \frac{1}{2}\sum_x \left| P_{X|\mathcal{E}}(x) - P_{Y|\mathcal{E}'}(x) \right| + \mathrm{Pr}[\overline{\mathcal{E}}] \cdot \frac{1}{2}\sum_x \left| P_{X|\overline{\mathcal{E}}}(x) - P_{Y|\overline{\mathcal{E}'}}(x) \right| \\
&\leq \mathrm{SD}\,(X|\mathcal{E}, Y|\mathcal{E}') + \mathrm{Pr}[\overline{\mathcal{E}}]\ .
\end{aligned}$$

□

3.2 Definition

In a model with idealized primitive P (chosen from some set $\mathcal{P}$), seedless extractors are algorithms $\mathrm{ext}^P : \mathcal{X} \to \mathcal{Y}$ with oracle access to P. The security definition for such extractors considers a two-stage attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, where both parts have access to P. The first stage $\mathcal{A}_1$ outputs a value x and some state information σ for $\mathcal{A}_2$. The second stage takes an input $y \in \mathcal{Y}$ and outputs a single bit (i.e., it acts as a distinguisher).

For an attacker $\mathcal{A}$, denote by $\mathcal{L}_1$ and $\mathcal{L}_2$ the (random variables corresponding to) the lists of the P-queries made by $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively.

Definition 3.5. An attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is called a *q-attacker* if $|\mathcal{L}_1 \cup \mathcal{L}_2| \leq q$ always; it is called a *q-IT-attacker* if $|\mathcal{L}_1| \leq q$ always.

That is, for IT-attackers the second stage $\mathcal{A}_2$ may make an arbitrary number of queries to P. Equivalently, $\mathcal{A}_2$ can be thought of as being given the entire function table of P.

The security game for seedless extractors in the P-model roughly requires that if the extractor is given a high-entropy input by $\mathcal{A}_1$, then $\mathcal{A}_2$ cannot tell the extractor output apart from a random value in $\mathcal{Y}$, even given the state information σ and access to P. Formally, it proceeds as follows:

1. The challenger chooses $b \leftarrow \{0,1\}$ and $P \leftarrow \mathcal{P}$ uniformly at random.
2. $\mathcal{A}_1$ gets access to P and produces $(\sigma, x) \leftarrow \mathcal{A}_1^P$.
3. The output of the extractor is computed as $y_0 \leftarrow \mathrm{ext}^P(x)$. Moreover, the challenger picks a value $y_1 \leftarrow \mathcal{Y}$ uniformly at random.
4. The second-stage attacker $\mathcal{A}_2$ is given σ and y_b and outputs a decision bit $b' \leftarrow \mathcal{A}_2^P(\sigma, y_b)$. The attacker wins if and only if $b' = b$.

The advantage of $\mathcal{A}$ in this extraction game is denoted by $\mathrm{Adv}_{\mathrm{ext}}^{\mathrm{ext},P}(\mathcal{A})$.

An attacker has to satisfy a legitimacy condition. Intuitively, this condition requires that the output X of $\mathcal{A}_1$ have high min-entropy even conditioned on the state information Σ and the list of queries $\mathcal{L}_1$.[2]

Definition 3.6. An attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is said to be γ^**-legitimate* if, in the extraction game above,

$$\mathrm{H}_\infty(X|\Sigma\mathcal{L}_1) \;\geq\; \gamma^* \,.$$

The above finally leads to the following definition of seedless extractor in the P-model:

Definition 3.7. An algorithm $\mathrm{ext}^P : \mathcal{X} \to \mathcal{Y}$ with oracle access to P is a seedless $(\gamma^*, q, \varepsilon)$*-(IT-)extractor in the P-model* if for every γ^*-legitimate q-(IT-)attacker $\mathcal{A}$,

$$\mathrm{Adv}_{\mathrm{ext}}^{\mathrm{ext},P}(\mathcal{A}) \;\leq\; \varepsilon \,.$$

3.3 Seedless Extraction with a Monolithic Random Oracle

For instructive purposes it is useful to consider monolithic extraction, i.e., the case where the ideal primitive P itself is used as an extractor. To exemplify this, assume P is a random oracle, i.e.,

[2] Note, in the extraction game the definition of $\mathcal{L}_1$ is the same in the real and the ideal worlds. For our future definitions of PRNGs, however, it will be important that the notion of legitimacy is defined in the ideal world (i.e., conditioned on $b = 1$).

a function $G : \{0,1\}^m \to \{0,1\}^n$ chosen uniformly at random. Then, the monolithic extractor is defined as follows:

Construction 1 (Monolithic extractor)**.** The monolithic seedless extractor $\mathsf{mono}^G : \{0,1\}^m \to \{0,1\}^n$ using a random oracle $G : \{0,1\}^m \to \{0,1\}^n$ is defined by

$$\mathsf{mono}^G(x) := G(x) \ .$$

Theorem 3.8 (Monolithic seedless extraction)**.** *Construction* mono *is a* $(\gamma^*, q, \varepsilon)$*-extractor in the* G*-model for*

$$\varepsilon \ \leq \ \frac{q}{2^{\gamma^*}} \ .$$

The proof of Theorem 3.8 is a straight-forward application of the H-coefficient technique. The idea is to first show that unless $\mathcal{A}_1$ or $\mathcal{A}_2$ queries the input x provided by $\mathcal{A}_1$, the real and ideal worlds (i.e., the cases where $b = 0$ and $b = 1$, respectively) are indistinguishable. That is, the corresponding ratio of transcript probabilities is 1. Transcripts where x is in the query list are defined to be *bad* transcripts, and the second part of the proof shows that bad transcripts are unlikely to occur due to the legitimacy of $\mathcal{A}$. The latter proof crucially relies on the fact that the H-coefficient technique enables performing the bad-event analysis in the *ideal* world.

Proof. Consider a transcript of the interaction between an attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and the challenger of the extraction game (as defined in Section 3.2). It consists of

- the input x provided by $\mathcal{A}_1$,
- the value y^* output by the game (which is either the output y_0 of the extractor on x or a uniformly random value from $\{0,1\}^n$), and
- the query/answer list $L = L_1 \cup L_2$ of $\mathcal{A}_1$'s and $\mathcal{A}_2$'s interaction with F.

That is $\tau = (y^*, x, L)$. A *bad* transcript occurs when $\mathcal{A}_1$ or $\mathcal{A}_2$ queries F at input x, i.e., when L contains a pair of the form $(x, *)$. In order to apply Theorem 3.1, one merely needs to bound the probability ratio for good transcripts (Lemma 3.9) and the probability of a bad transcript occurring in the ideal world, i.e, for $b = 1$ (Lemma 3.10). □

Lemma 3.9 (Ratio analysis)**.** *For all* good *transcripts* τ*,*

$$\frac{\mathrm{p}_0(\tau)}{\mathrm{p}_1(\tau)} \;=\; 1\,.$$

Proof. Fix a good transcript τ and consider first $\mathrm{p}_1(\tau)$. Since in the ideal world y^* is sampled uniformly,

$$\mathrm{p}_1(\tau) \;=\; p_L \cdot 2^{-n}\,,$$

where p_L denotes the probability that a uniform random function is consistent with the queries in L. In the real world,

$$\mathrm{p}_0(\tau) \;=\; p_L \cdot q_\tau\,,$$

where q_τ is the probability that $F_L(x) = y^*$ over a function F_L that is sampled uniformly at random conditioned on being consistent with L. Since τ is a good transcript, F_L is not constrained by L at coordinate x, and, hence, $q_\tau = 2^{-n}$. □

Remark 1. In the above proof, note that p_L does not include the probability that x appears in the transcript. Recall from Section 3.1.1 that the behaviors $\mathrm{p}(\tau)$ are the probabilities that the experiment produces the answers in τ *when given* the queries in τ (and the value x is a query by the distinguisher).

Lemma 3.10 (Bad event analysis)**.** *For the set* $\mathcal{B}$ *of bad transcripts (as defined above),*

$$\Pr[T_1 \in \mathcal{B}] \;\leq\; \frac{q}{2^{\gamma^*}}\,.$$

Proof. Recall that by the γ^*-legitimacy of $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$,

$$\mathrm{H}_\infty(X|\Sigma\mathcal{L}_1) \;\geq\; \gamma^* \,.$$

Observe that in the ideal world, the output of the extraction game is a uniformly random value Y^*, which is independent of the input X produced by $\mathcal{A}_1$. The sampling order of the ideal experiment can therefore be changed to be the following:

1. Sample F uniformly at random.
2. Run $\mathcal{A}_1$ until it outputs σ and x, thereby also generating the list of queries L_1.
3. Choose y^* uniformly at random.
4. Run $\mathcal{A}_2$ on input (σ, y^*), letting it make additional queries L_2.
5. Resample the input X conditioned on $(\Sigma, \mathcal{L}_1) = (\sigma, L_1)$.

Note that since the conditioning includes $\mathcal{L}_1$, $\mathcal{A}_1$ makes the same queries, L_1, during the first run and the resampling process. Moreover, since conditioned on the values of $(\Sigma, \mathcal{L}_1)$, X and $\mathcal{L}_2$ are independent, the min-entropy condition holds for $\mathcal{L} = \mathcal{L}_1 \cup \mathcal{L}_2$, the list of all queries made by $\mathcal{A}$ during the experiment, as well. That is,

$$\mathrm{H}_\infty(X|\Sigma\mathcal{L}) \;\geq\; \gamma^* \,.$$

Thus, the probability that the resampled input X is contained in any query in the list L is at most $q \cdot 2^{-\gamma^*}$. □

Parameter choices. In terms of concrete parameters, observe the following for the constructions towards monolithic seedless extraction from above:

- **Computational:** If we let $n = 512$ and $q = 2^{80}$. We would need $\gamma^* \approx 160$ to get 80 bits of security.

- **Information Theoretic:** We let $n = 512$. We also approximate $1/(1-\rho) \leq 2$, very generously Then, if we set for example $q = 2^{80}$. We would need the entropy loss, i.e, $\gamma^* = 160$ for 80 bits of security.

Theorem 3.11 (Monolithic seedless IT-extraction)**.** *Construction* mono *is a* $(\gamma^*, q, \varepsilon)$-IT-extractor in the G-model *for*

$$\varepsilon \leq \frac{1}{2}\sqrt{\frac{2^{-(\gamma^*-n)}}{1-\rho}} + \rho \,,$$

where $\rho = q/2^{\gamma^*}$.

The proof of Theorem 3.11 proceeds by bounding the statistical distance of $\mathcal{A}_2$'s views in the real and ideal experiments via the corresponding collision probabilities (as done in the proof of the left-over hash lemma). In the proofs of the actual PRNG constructions in the following sections, bounding said collision probabilities constitutes the bulk of the proof and is quite involved.

Proof. Consider the extraction game corresponding to Definition 3.7 (cf. Section 3.2). Since $\mathcal{A}_2$ gets to make an unbounded number of queries to the random oracle G, one may equivalently consider the game where $\mathcal{A}_2$ simply gets the entire function table of G as input. Therefore, the distinguishing advantage of $\mathcal{A}_2$ is upper bounded by

$$\mathrm{SD}\left((\Sigma, Y_0, \mathcal{L}_1, G), (\Sigma, Y_1, \mathcal{L}_1, G)\right) \,,$$

where Σ is (the random variable corresponding to) the state information output by $\mathcal{A}_1$, $Y_0 = G(X)$, Y_1 a uniform random string, and $\mathcal{L}_1$ the query/answer list by $\mathcal{A}_1$.

Let $\mathcal{E}$ be the event that $\mathcal{A}_1$ does not query the value X it provides as input to the extractor. Observe that this event can be defined in both the real ($b = 0$) and ideal ($b = 1$) experiments. In the ideal experiment, the probability of $\mathcal{E}$ not occurring is easy to bound: Since Y_1 is uniformly random and independent of X, consider the following equivalent way of sampling a tuple $(\Sigma, X, Y_1, \mathcal{L}_1, G)$:

1. Sample Y_1 and G uniformly at random.
2. Run $\mathcal{A}_1^G$ to produce Σ, $\tilde{X}$, and $\mathcal{L}_1$.
3. Rerun $\mathcal{A}_1^G$ with fresh randomness, but conditioned on the state information being Σ and the queries being $\mathcal{L}_1$. This results in a new value X.
4. Output $(\Sigma, X, Y_1, \mathcal{L}_1, G)$.

It is easily seen that the distribution produced via resampling is identical to that of the actual experiment.

Note that for particular values σ and L_1,

$$\Pr[\overline{\mathcal{E}}|\sigma L_1] \;\leq\; q \cdot \mathsf{Pred}(X|\sigma L_1)\,,$$

which can be easily seen due to the alternative sampling above. Hence, taking expectations,

$$\Pr[\overline{\mathcal{E}}] \;\leq\; q \cdot \mathsf{Pred}(X|\Sigma\mathcal{L}_1) \;\leq\; q \cdot 2^{-\gamma^*}\,,$$

using the γ^*-legitimacy of $\mathcal{A}$ in the last step.

Using Proposition 3.4, one obtains

$$\mathsf{SD}\left((\Sigma, Y_0, \mathcal{L}_1, G), (\Sigma, Y_1, \mathcal{L}_1, G)\right) \;\leq\; \mathsf{SD}\left((\Sigma, Y_0, \mathcal{L}_1, G)|\mathcal{E}, (\Sigma, Y_1, \mathcal{L}_1, G)|\mathcal{E}\right) + q \cdot 2^{-\gamma^*}\,.$$

In order to bound the statistical distance conditioned on $\mathcal{E}$, condition additionally on arbitrary values $z = (\sigma, L_1)$ (with non-zero probability given $\mathcal{E}$). Observe that under such conditioning, G is chosen uniformly at random from all functions consistent with G. Using Proposition 3.3, one bounds the desired statistical distance as

$$\mathsf{SD}\left((Y_0, G)|z\mathcal{E}, (Y_1, G)|z\mathcal{E}\right) \;\leq\; \frac{1}{2}\sqrt{2^n \cdot \mathsf{Coll}(Y_0|Gz\mathcal{E}) - 1}\,.$$

To bound the collision probability $\mathsf{Coll}(Y|Gz\mathcal{E})$, consider the following experiment:[3]

1. Sample G uniformly at random consistent with L_1.
2. Sample inputs $X \leftarrow \mathcal{A}_1^G$ and $X' \leftarrow \mathcal{A}_1^G$ independently but conditioned on $Z = z$ and $\mathcal{E}$.
3. Compute $Y_0 \leftarrow \mathsf{mono}^G(X)$ and $Y_0' \leftarrow \mathsf{mono}^G(X')$ respectively.

The collision probability $\mathsf{Coll}(Y|Gz)$ is therefore equal to

$$\Pr[Y_0 = Y_0'] \;\leq\; \Pr[X = X'] + \Pr[Y = Y'|X \neq X']$$

in the experiment above. The former term is at most $p_z := \mathsf{Pred}(X|Z = z, \mathcal{E})$. For the latter term, consider arbitrary $x \neq x'$. Since neither x nor x' is covered by L_1, $Y_0 = Y_0'$ occurs with probability 2^{-n}. Hence,

$$\mathsf{SD}\left((Y_0, G)|z\mathcal{E}, (Y_1, G)|z\mathcal{E}\right) \;\leq\; \frac{1}{2}\sqrt{2^n(p_z + 2^{-n}) - 1} \;\leq\; \frac{1}{2}\sqrt{2^n p_z}\,.$$

Using Jensen's inequality, one obtains

$$\begin{aligned}\mathsf{SD}\left((Y_0, G)|z\mathcal{E}, (Y_1, G)|z\mathcal{E}\right) \;&\leq\; \frac{1}{2}\sqrt{2^n \cdot \mathsf{Pred}(X|Z\mathcal{E})} \\ &\leq\; \frac{1}{2}\sqrt{\frac{2^{-(\gamma^* - n)}}{(1-\rho)}}\,,\end{aligned}$$

[3] Observe that, in general, $\mathsf{Coll}(U|V)$ is equal to the probability that $U = U'$ in the experiment where one jointly samples U and V, and then resamples U' conditioned on the value of V.

where the last inequality follows from

$$\begin{aligned}\mathsf{Pred}(X|Z) &\geq \sum_z \Pr[Z = z \wedge \mathcal{E}] \cdot p_z \\ &= \Pr[\mathcal{E}] \cdot \sum_{z \in \mathcal{E}} \frac{\Pr[Z = z \wedge \mathcal{E}]}{\Pr[\mathcal{E}]} \cdot p_z \\ &= \Pr[\mathcal{E}] \cdot \mathsf{Pred}(\overline{X}|Z\mathcal{E}) \ .\end{aligned}$$

and $\mathsf{Pred}(X|Z) \leq 2^{-\gamma^*}$, by the legitimacy of $\mathcal{A}$. □

3.4 Online Extraction

In practice it is uncommon to assume access to a monolithic random oracle. Instead, practical hash functions are usually built from (public) compression functions, ciphers, or permutations. These underlying primitives P have limited input length and will therefore not be able to process inputs of arbitrary length m. Therefore, extractors (and PRNGs) should be designed in such a way that they can process short m-bit input blocks (e.g., $m = 256, 512, 1600$) and accumulate their entropy in the internal state.

An "accumulating" extractor satisfies ext satisfies the same security Definition 3.7, but its syntax can be thought of as two algorithms $\mathsf{ext} = (\mathsf{refresh}, \mathsf{finalize})$, where refresh accumulates entropy in an internal state and finalize produces the extractor output from the current state.

Definition 3.12. An *online extractor construction* consists of two algorithms $\mathsf{ext} = (\mathsf{refresh}, \mathsf{finalize})$, where

- refresh takes a state s and an input $x \in \{0,1\}^m$ and produces a new state $s' \leftarrow \mathsf{refresh}^P(s, x)$, and
- finalize takes a state s and produces an output $y \in \{0,1\}^r$, i.e., $y \leftarrow \mathsf{finalize}^P(s)$.

An online extractor processing m-bit inputs and producing r-bit output is called a (m, r)-*online extractor*.

The security definition for online extractors additionally considers the number ℓ of times refresh is called by the attacker, i.e., it considers (q, ℓ)-attackers.

Definition 3.13. An algorithm $\mathsf{ext}^P : \mathcal{X} \to \mathcal{Y}$ defined by two algorithms $\mathsf{ext} = (\mathsf{refresh}, \mathsf{finalize})$ with oracle access to P is an $(\gamma^*, q, \ell, \varepsilon)$-*(IT-)online extractor in the P-model* if for every γ^*-legiti -mate (q, ℓ)-(IT-)attacker $\mathcal{A}$,

$$\mathrm{Adv}^{\mathsf{ext},P}_{\mathsf{ext}}(\mathcal{A}) \;\leq\; \varepsilon\,.$$

Now, we will present the constructions of such online extractors where such online extractors can be obtained from Merkle-Damgård with a random compression function, from Merkle-Damg ård with the Davies-Meyer compression function, and from Sponges. Indeed, PRNGs can be built much like these online extractors and will be discussed in subsequent chapters. In addition, we will also consider the HMAC construction as a seedless extractor. HMAC is roughly based on Merkle-Damgård, but it has a few modifications/additions that are unnecessary for extraction. However, due to its wide-spread use, we point out how to modify the Merkle-Damgård proofs to obtain a security statement for HMAC. These constructions are pictorially represented in Figure 3.1.

Finally, we will also look at the extractor built using the CBC paradigm (which can be thought of as an "extreme sponge") and mount an attack to show that such a construction does not lead to a secure online extractor.

3.5 Constructions of Online Extractors

This section presents three simple, intuitive, and —most importantly— practical online-extrac -tor constructions:

- a construction based on the *Merkle-Damgård paradigm* using a public *fixed-length compression function*;
- a construction based on the *Merkle-Damgård paradigm* using the *Davies-Meyer compression function* (as in SHA-2), which is built from any public block cipher; and
- a construction based on the *Sponge paradigm* (as in SHA-3), which uses a public permutation.

For extractors based on the MD paradigm, there are in fact two constructions: one achieving normal, computational security and one achieving information-theoretic (IT) security. As with PRNGs, there is also an IT candidate based on Sponges, but its security analysis is left for future work.

3.5.1 Extractors from Merkle-Damgård

An online extractor can be obtained from a compression function F as follows:[4]

Construction 2 (Online extractor from Merkle-Damgård)**.** The (m, n)-online extractor construction $\mathsf{MD} = (\mathsf{refresh}, \mathsf{finalize})$ based on Merkle-Damgård with a compression function $F : \{0,1\}^n \times \{0,1\}^m \longrightarrow \{0,1\}^n$ is defined as follows:

- $\mathsf{refresh}^F(s, x) = F(s, x)$, and
- $\mathsf{finalize}^F(s) = s$.

The security of Construction 2 is proved in the F-model, where F is a uniformly random function. The theorem below is identical to Lemma 5.2.

[4] To reduce notational clutter, the algorithms refresh and finalize of the extractor constructions are not "branded" with the design name. There will be no ambiguity as to which construction is meant in any place in this paper.

Theorem 3.14 (Online extractor from Merkle-Damgård)**.** *Construction 2 is a* $(\gamma^*, q, \varepsilon)$*-online extractor in the* F*-model for*

$$\varepsilon \leq \frac{q^2 + q\ell + \ell^2}{2^n} + \frac{2q}{2^{\gamma^*}} .$$

An IT-secure online extractor based on Merkle-Damgård can be obtained if the finalize function simply truncates the state:

Construction 3 (IT online extractor from Merkle-Damgård)**.** The (m, r)-IT-online extractor construction $\mathsf{MD}_r = (\mathsf{refresh}, \mathsf{finalize})$ based on Merkle-Damgård with a compression function $F : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ is defined as follows:

- $\mathsf{refresh}^F(s, x) = F(s, x)$, and
- $\mathsf{finalize}^F(s) = s[1..r]$.

The security of Construction 3 is proved in the F-model, where F is a uniformly random function. To state the theorem for the IT construction, for an integer ℓ, let

$$d'(\ell) = \max_{\ell' \in \{1,\ldots,\ell\}} |\{d \in \mathbb{N} \ : \ d | \ell'\}| .$$

Observe that, asymptotically, $d'(\ell)$ grows very slowly, i.e., as $\ell^{o(1)}$. Furthermore, let F be a random compression function. The following theorem is equivalent to Lemma 5.10.

Theorem 3.15 (IT online extractor from Merkle-Damgård)**.** *Construction 3 is a* $(\gamma^*, q, \varepsilon)$*-IT-online extractor in the* F*-model, where*

$$\varepsilon \leq \frac{1}{2}\sqrt{\frac{2^{r-\gamma^*}}{(1-\rho)} + \ell \cdot d'(\ell) \cdot \frac{2^r}{2^n} + 64\ell^4 \cdot \frac{2^r}{2^{2n}} + 16\ell^2 \frac{q^2 2^r}{2^{2n}}} + \rho \ ,$$

where $\rho = \frac{q^2}{2^r}$.

3.5.2 Extractors from Merkle-Damgård with Davies-Meyer

The Davies-Meyer compression function maps two inputs $a \in \{0,1\}^m$ and $b \in \{0,1\}^n$ to an n-bit string

$$E(b,a) \oplus a\,,$$

where E is an arbitrary block cipher (where b is the key and a the input).[5] Correspondingly, an online extractor can be obtained from E as follows:

Construction 4 (Online extractor from MD-DM)**.** The (k,n)-online extractor construction $\mathsf{DM} = (\mathsf{refresh}, \mathsf{finalize})$ based on Merkle-Damgård with Davies-Meyer (MD-DM) uses a cipher $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ and is defined as follows:

- $\mathsf{refresh}^E(s,x) = E(x,s) \oplus s$, and
- $\mathsf{finalize}^F(s) = s$.

The security of Construction 4 is proved in the E-model, where E is a cipher chosen uniformly at random from the set of all ciphers and can be queried in both the forward and backward direction. The theorem below is identical to Lemma 5.19

Theorem 3.16 (Online extractor from MD-DM)**.** *Construction 4 is a $(\gamma^*, q, \varepsilon_{\text{rob}})$-robust online extractor in the E-model for*

$$\varepsilon_{\text{rob}} \;\le\; \frac{q^2 + 2(q\ell + \ell^2)}{2^n} + \frac{4q}{2^{\gamma^*}}\,.$$

An IT-secure online extractor based on MD-DM can be obtained if the finalize function simply truncates the state:

Construction 5 (IT online extractor from MD-DM)**.** The (k,r)-IT-online construction $\mathsf{DM}_r = (\mathsf{refresh}, \mathsf{finalize})$ using Merkle-Damgård with Davies-Meyer (MD-DM) uses a block cipher $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ and is defined as follows:

[5] A (block) cipher is an efficiently computable and invertible permutation $E(k,\cdot) : \{0,1\}^n \to \{0,1\}^n$ for every key $k \in \{0,1\}^n$.

- $\mathsf{refresh}^E(s, x) = E(x, s) \oplus s$, and
- $\mathsf{finalize}^E(s) = s[1..r]$.

The security of Construction 5 is proved in the E-model, where E is a cipher chosen uniformly at random from the set of all ciphers and can be queried in both the forward and backward direction. Let $d'(\ell)$ be defined as in Section 5.1. The following theorem is equivalent to Lemma 5.27.

Theorem 3.17 (IT online extractor from MD-DM)**.** *Construction 5 is a $(\gamma^*, q, \varepsilon_{\text{rob}})$-IT-online extractor in the E-model, where*

$$\mathrm{Adv}^{\text{rec-IT},\gamma^*}_{\mathsf{DM}}(\mathcal{A}) \leq \frac{1}{2}\sqrt{\frac{2^{r-\gamma^*}}{(1-\rho)} + \ell \cdot d'(\ell)\frac{2^r}{2^{n-1}} + \frac{64\ell^4 2^r}{2^{2n-2}} + \frac{16\ell^2 q^2 2^r}{2^{2n-2}}} + \rho\ .$$

where $\rho = \frac{q^2}{2^r}$.

3.5.3 Extractors from Sponges

Let $n \in \mathbb{N}$ and $n = r+c$. In the following, for an n-bit string s, let $s = s^{(\mathrm{r})} \| s^{(\mathrm{c})}$ be decomposition of s into an r-bit and c-bit string. An online extractor using the Sponge paradigm can be obtained from a permutation π as follows:

Construction 6 (Online extractor from Sponges)**.** The Sponge-based online-extractor construction $\mathsf{Spg} = (\mathsf{refresh}, \mathsf{finalize})$ uses a permutation $\pi : \{0,1\}^n \to \{0,1\}^n$ to absorb and produce r-bit inputs and outputs, respectively, and is defined as follows:

- $\mathsf{refresh}^\pi(s, x) = \pi(s \oplus x \| 0^c)$, and
- $\mathsf{finalize}^\pi(s) = s[1..r]$.

Observe that for Sponge-based extractors, even the computational variant needs to truncate the state, otherwise the output of the extractor could be inverted by the attacker, which renders the constructions insecure.

The security of Construction 6 is proved in the π-model, where π is a uniformly random permutation, which can be queried in both the forward and backward direction. The proof of the following theorem follows along similar lines as that of Lemma 5.36.

Theorem 3.18 (Online extractor from Sponges)**.** *Construction 6 is a $(\gamma^*, q, \ell, \varepsilon)$-online extractor in the π-model for*

$$\varepsilon \;\leq\; 2 \cdot \left(\frac{q + q\ell + \ell^2}{2^n} + \frac{q^2}{2^c} + \frac{q}{2^{\gamma^*}} \right) .$$

Observe that the bound in Theorem 5.30 is only reasonable when c is large enough, which matches the fact that CBC-based online extractors—which correspond to the case $c = 0$, are not secure.

As pointed out before, the IT-secure online extractor based on Sponge would be similar to the computational variant. We reproduce the definition for completeness.

Construction 7 (IT Online extractor from Sponges)**.** The Sponge-based IT-online-extractor construction Spg = (refresh, finalize) uses a permutation $\pi : \{0,1\}^n \to \{0,1\}^n$ to absorb and produce r-bit inputs and outputs, respectively, and is defined as follows:

- $\mathsf{refresh}^{\pi}(s, x) = \pi(s \oplus x \| 0^c)$, and
- $\mathsf{finalize}^{\pi}(s) = s[1..r]$.

The security of Construction 7 is proved in the π-model, where π is an ideal permutation chosen uniformly at random from the set of all permutations and can be queried in both the forward and backward direction. The following theorem is equivalent to Lemma 5.39.

Theorem 3.19 (IT online extractor from Sponges)**.** *Construction 7 is a $(\gamma^*, q, \varepsilon_{\mathrm{rob}})$-IT-online extractor in the π-model, where*

$$\mathrm{Adv}^{\text{rec-IT},\gamma^*}_{\mathsf{Spg}}(\mathcal{A}) \;\leq\; \frac{1}{2}\sqrt{\frac{2^{r-\gamma^*}}{(1-\rho)} + \frac{\ell \cdot (\ell + q)}{2^{c-1}}} + \rho \; .$$

where $\rho = \frac{q^2}{2^c}$.

3.5.4 Extractors from HMAC

In practice, uniformly random key material is often derived from high-entropy inputs (resulting, e.g., from a key-agreement protocol) using a *key-derivation function (KDF)*. A common paradigm, suggested by Krawczyk [Kra10], to construct KDFs is to combine an extractor with a variable-length pseudorandom function (VL-PRF). The most widely used KDF is *HKDF*, which uses the HMAC mode of operation for compression functions (CF) to instantiate both the extractor and the VL-PRF. This section considers the security of HMAC as a *seedless extractor* w.r.t. the new legitimacy condition put forth by this work. Together with a VL-PRF, the seedless HMAC extractor can then be used to build a KDF. A full treatment of seedless KDFs is deferred to future work.

HMAC. HMAC is similar to Merkle-Damgård, but requires additional CF calls, designed to prevent extension attacks when HMAC is used as a PRF. Concretely, for a compression function $F : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$, the HMAC construction takes a key $k \in \{0,1\}^m$ as well as inputs $x_1, \ldots, x_\ell \in \{0,1\}^m$ and outputs[6]

$$\mathsf{HMAC}(k, x_1, \ldots, x_\ell) \;:=\; \mathsf{MD}^F(k \oplus \mathsf{opad}, \mathsf{MD}^F(k \oplus \mathsf{ipad}, x_1, \ldots, x_\ell)) \,,$$

where $\mathsf{ipad} \neq \mathsf{opad} \in \{0,1\}^m$ are arbitrary constants. HMAC can be used as seedless extractor by fixing, say, $k := 0$. That is, the online extractor based on HMAC would be defined as follows:

Construction 8 (Online extractor from HMAC.)**.** The (m, n)-online extractor construction $\mathsf{HMAC} = (\mathsf{refresh}, \mathsf{finalize})$ based on HMAC with a compression function $F : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ is defined as follows:

- the initial state is $s_0 = F(0, \mathsf{ipad})$;

[6] The IV to the Merkle-Damgård construction is 0.

- $\mathsf{refresh}^F(s, x) = F(s, x)$; and
- $\mathsf{finalize}^F(s) = F(F(0, \mathrm{opad}), s)$.

The security of Construction 2 is proved in the F-model, where F is a uniformly random function.

Theorem 3.20 (Online extractor from HMAC)**.** *Construction 8 is a $(\gamma^*, q, \varepsilon)$-online extractor in the F-model for*

$$\varepsilon \leq \frac{q^2 + q + q\ell + \ell^2}{2^n} + \frac{2q}{2^{\gamma^*}} .$$

Proof (sketch). To prove the security of HMAC as an extractor according to Definition 3.7, one first considers a hybrid experiment, in which the output of the extractor is computed as

$$F(F(0, \mathrm{opad}), U) ,$$

for a value chosen uniformly at random. The indistinguishability of the original extraction game and the hybrid experiment is established via a simple hybrid argument: essentially, the reduction (to the security of MD as an extractor) simply prepends a block ipad to the inputs $x_1, \ldots, x_\ell$, and upon receiving a value y, it additionally computes $F(F(0, \mathrm{opad}), y)$. It is easily seen that adding ipad to the input does not affect the conditional entropy of the input—and therefore the legitimacy of the reduction.

Finally, it is easily seen that in the hybrid experiment, the advantage of any attacker is zero unless it queries $F(F(0, \mathrm{opad}), U)$, which happens with probability at most $q/2^n$. □

An IT-secure online extractor based on Merkle-Damgård can be obtained if the finalize function additionally truncates the output:

Construction 9 (IT online extractor from HMAC.)**.** The (m, r)-IT-online extractor construction $\mathsf{HMAC}' = (\mathsf{refresh}, \mathsf{finalize})$ based on HMAC with a compression function $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is defined as follows:

- the initial state is $s_0 = F(0, \text{ipad})$;
- $\text{refresh}^F(s, x) = F(s, x)$; and
- $\text{finalize}^F(s) = F(F(0, \text{opad}), s)[1..r]$.

The security of Construction 9 is proved in the F-model, where F is a uniformly random function. To state the theorem for the IT construction, for an integer ℓ, let

$$d'(\ell) = \max_{\ell' \in \{1,\ldots,\ell\}} |\{d \in \mathbb{N} \ : \ d | \ell'\}| \,.$$

Observe that, asymptotically, $d'(\ell)$ grows very slowly, i.e., as $\ell^{o(1)}$. Furthermore, let F be a random compression function. The following theorem is equivalent to Lemma 5.10.

Theorem 3.21 (IT online extractor from HMAC)**.** *Construction 9 is a $(\gamma^*, q, \varepsilon)$-IT-online extractor in the F-model, where*

$$\varepsilon \leq \frac{1}{2}\sqrt{\frac{2^{r-\gamma^*}}{(1-\rho)} + \ell \cdot d'(\ell) \cdot \frac{2^r}{2^n} + 64\ell^4 \cdot \frac{2^r}{2^{2n}} + 16\ell^2 q^2 \cdot \frac{2^r}{2^{2n}} + (q + \ell + 2) \cdot \frac{2^r}{2^n}} + \rho \ ,$$

where $\rho = \frac{q^2}{2^n}$.

Proof. We define a few random variables which we will be using in our proofs.

- F a randomly chosen compression function, to which the adversary is given access. We use F both for the oracle itself, as well as for the random variable describing the entire function table.
- ℓ: Number of blocks input to the challenge oracle (which is a random variable itself, we overload notation here, using the same letter we use in the bound on ℓ in the lemma statement).
- $\overline{X} = (\overline{X}_1, \ldots, \overline{X}_\ell)$: the blocks input to the challenge oracle.

- $\tilde{Y}_\ell$: output of HMAC′. Let us remind ourselves that this is s truncated to r bits (the output of finalize);
- $Z = (\Sigma, \mathcal{L}, S_0)$: "side information" where Σ is the attacker state before challenge, $\mathcal{L}$ is the attacker query/answers to F before challenge, S_0 is the initial state provided by $\mathcal{A}$;
- U_r: uniform r-bit string.
- S: the state of the extractor. For the purposes of this proof, we will be using S to indicate the state of the extractor that is the input to finalize.

The advantage of the adversary $\mathcal{A}$ in the online extractor game is bounded by $\mathsf{SD}\left((\tilde{Y}_\ell, Z, F), (U_r, Z, F)\right)$. Therefore, it is sufficient to upper-bound just that. The reason follows from the fact that Z contains the state of the attacker Σ just before it makes the challenge query. This means that $\mathcal{A}$ cannot tell apart real from random.

Much like the earlier proofs, we define an event $\mathcal{E}$ where the answers to the F-queries by $\mathcal{A}$ are distinct *when truncated to the first r bits*. Note that there are a maximum of q queries in this list. As pointed out earlier $\mathcal{E}$ has the same probability of occurring in either experiment, since the experiments are identical up to the point when this event is defined. Therefore, by Proposition 3.4,

$$\mathsf{SD}\left((\tilde{Y}_\ell, Z, F), (U_r, Z, F)\right) \;\leq\; \mathsf{SD}\left((\tilde{Y}_\ell, Z, F)|\mathcal{E}, (U_r, Z, F)|\mathcal{E}\right) + \frac{q^2}{2^r}\,;$$

For convenience we let $\rho := q^2/2^r$ for the remainder of the proof. In order to bound the statistical distance conditioned on $\mathcal{E}$, we can rewrite the same as as

$$\mathsf{SD}\left((\tilde{Y}_\ell, Z, F)|\mathcal{E}, (U_r, Z, F)|\mathcal{E}\right) \;=\; \sum_{z \in \mathcal{E}} \Pr[Z = z|\mathcal{E}] \cdot \mathsf{SD}\left((\tilde{Y}_\ell, F)|z, (U_r, F)|z\right)\,, \tag{3.2}$$

where $z \in \mathcal{E}$ is to denote that the sum is taken over all side informations $Z = z$, satisfying $\mathcal{E}$.[7] Define $p_z := \mathsf{Pred}(\overline{X}|Z = z)$, and observe that

$$\mathbf{E}_z[p_z] \;=\; \mathsf{Pred}(\overline{X}|Z) \;\leq\; 2^{-\gamma^*}\,,$$

where the latter inequality follows from the assumption $\mathrm{H}_\infty(\overline{X}|Z) \geq \gamma^*$. Moreover,

$$\mathrm{H}_\infty(\overline{X}|Z\mathcal{E}) \;\geq\; \gamma^* - \log(1-\rho)^{-1}\,,$$

which is due to

$$\begin{aligned}
\mathsf{Pred}(\overline{X}|Z) \;&\geq\; \sum_{z\in\mathcal{E}} \Pr[Z=z]\cdot p_z \\
&=\; \Pr[\mathcal{E}]\cdot \sum_{z\in\mathcal{E}} \frac{\Pr[Z=z]}{\Pr[\mathcal{E}]}\cdot p_z \\
&=\; \Pr[\mathcal{E}]\cdot \mathsf{Pred}(\overline{X}|Z\mathcal{E})\,.
\end{aligned}$$

From Lemma 3.22 we prove below, we will get,

$$\mathsf{SD}\left((\tilde{Y}_\ell, F)|z, (U_r, F)|z\right) \;\leq\; \frac{1}{2}\sqrt{2^r p_z + \ell\cdot d'(\ell)\cdot\frac{2^r}{2^n} + 64\ell^4\cdot\frac{2^r}{2^{2n}} + 16\ell^2q^2\cdot\frac{2^r}{2^{2n}} + (q+\ell+2)\cdot\frac{2^r}{2^n}}\;.$$

Using Jensen's inequality, (3.2) becomes, for $\alpha = 2^r$ and
$\beta = \ell\cdot d'(\ell)\cdot\frac{2^r}{2^n} + 64\ell^4\cdot\frac{2^r}{2^{2n}} + 16\ell^2q^2\cdot\frac{2^r}{2^{2n}} + (q+\ell+1)\cdot\frac{2^r}{2^n}$,

$$\begin{aligned}
\mathsf{SD}\left((\tilde{Y}_\ell, F)|\mathcal{E}, (U_r, F)|\mathcal{E}\right) \;&\leq\; \frac{1}{2}\sqrt{\alpha\mathsf{Pred}(\overline{X}|\mathcal{L}\mathcal{E}) + \beta} \\
&\leq \frac{1}{2}\sqrt{\frac{2^{r-\gamma^*}}{(1-\rho)} + \ell\cdot d'(\ell)\cdot\frac{2^r}{2^n} + 64\ell^4\cdot\frac{2^r}{2^{2n}} + 16\ell^2q^2\cdot\frac{2^r}{2^{2n}} + (q+\ell+2)\cdot\frac{2^r}{2^n}}\;.
\end{aligned}$$

[7] Therefore $\mathcal{E}$ can be omitted in the conditioning of the statistical distance.

□

Lemma 3.22. *For $z \in \mathcal{E}$ and the random variables as defined earlier,*

$$\mathsf{SD}\left((\tilde{Y}_\ell, F)|z, (U_r, F)|z\right) \;\leq\; \frac{1}{2}\sqrt{2^r p_z + \ell \cdot d'(\ell) \cdot \frac{2^r}{2^n} + 64\ell^4 \cdot \frac{2^r}{2^{2n}} + 16\ell^2 q^2 \cdot \frac{2^r}{2^{2n}} + (q+\ell+2) \cdot \frac{2^r}{2^n}} \;.$$

Proof. Fix $z = (\sigma, L, s_0)$. When we condition on z, it becomes evident that F is uniformly distributed over all functions that agree with L. We now use Proposition 3.3 to get that:

$$\mathsf{SD}\left((\tilde{Y}_\ell, F)|z, (U_r, F)|z\right) \;\leq\; \frac{1}{2}\sqrt{2^r \cdot \mathsf{Coll}(\tilde{Y}_\ell|Fz) - 1} \;. \tag{3.3}$$

To bound the collision probability, we consider the following experiment:

- choose F uniformly consistent with L
- sample inputs $\overline{X} = (\overline{X}_1, \ldots, \overline{X}_\ell)$ and $\overline{X}' = (\overline{X}'_1, \ldots, \overline{X}'_{\ell'})$ independently but conditioned on $Z = z$.
- compute S, S' as the refresh evaluations with F of inputs $\overline{X}$ and $\overline{X}'$ respectively.
- compute $\tilde{Y}_\ell$ and $\tilde{Y}'_\ell$ as the truncated HMAC′ evaluations with F of $\overline{X}$ and $\overline{X}'$, i.e $\tilde{Y}_\ell = F(F(0, \mathsf{opad}), S)[1..r]$ and $\tilde{Y}'_\ell = F(F(0, \mathsf{opad}), S')[1..r]$

We first condition on the event $\mathcal{E}$ that $F(0, \mathsf{opad}) \neq F(0, \mathsf{ipad})$. Then, we bound the probability that $\tilde{Y}_\ell = \tilde{Y}'_\ell$ in this modified experiment which is conditioned on $\mathcal{E}$ as

$$\Pr[\tilde{Y}_\ell = \tilde{Y}'_\ell] \;\leq\; \frac{1}{2^n} + \Pr[\overline{X} = \overline{X}'] + \Pr[\tilde{Y}_\ell = \tilde{Y}'_\ell | \overline{X} \neq \overline{X}'][8] \;. \tag{3.4}$$

Now, the first term reduces to at most p_z. Towards bounding the second term, we fix arbitrary inputs $\overline{x} \neq \overline{x}'$ of lengths ℓ and ℓ', respectively. We also assume, wlog, that the evaluation of $\overline{x}'$ is not completely covered by L; otherwise it would violate the collision-freeness of L. Let $\overline{x}_{k+1}$

be the first block of $\overline{x}$ not covered by L and similarly $\overline{x}'_{k+1}$ for $\overline{x}'$. We let $k = \ell$ if all blocks are covered. We now use the results of Lemma 3.23 to upperbound $\Pr[\tilde{Y}_\ell = \tilde{Y}'_\ell]$. This concludes the proof. □

Lemma 3.23. *For fixed inputs $\overline{x} \neq \overline{x}'$ and the random variables as defined earlier,*

$$\Pr[\tilde{Y}_\ell = \tilde{Y}'_\ell] \leq \frac{\ell \cdot d'(\ell)}{2^n} + \frac{64\ell^4}{2^{2n}} + \frac{16\ell^2 q^2}{2^{2n}} + \frac{q+\ell+1}{2^n} + \frac{1}{2^r} .$$

Proof. Upon fixing it to be arbitrary inputs, we can drop the conditioning on $\overline{X} \neq \overline{X}'$. In this setting, we can condition on the equality of S and S' as follows:

$$\Pr[\tilde{Y}_\ell = \tilde{Y}'_\ell] \leq \Pr[S = S'] + \Pr[\tilde{Y}_\ell = \tilde{Y}'_\ell | S \neq S'] . \tag{3.5}$$

Let us take a look at the first term. Notice that this is the collision probability of S when run on two fixed inputs $\overline{x}$ and $\overline{x}'$. In other words, this proof is similar to the bounding of collision probability of truncated outputs of MD when run on $\overline{X}$ with side information z. We now proceed to construct the structure graph as done in the Proof of Lemma 5.11. The key difference here is that we need the structure graph to be colliding, i.e $\Pr[S = S'] = \Pr[G_F(\overline{x}, \overline{x}') \in \mathsf{Coll}_L(\overline{x}, \overline{x}')]$ where $\mathsf{Coll}_L(\overline{x}, \overline{x}')$ are the set of colliding structure graphs. We now use the results of Lemmas 5.15, 5.16 and 5.17 to conclude that:

$$\Pr[S = S'] \leq \frac{\ell \cdot d'(\ell)}{2^n} + \frac{64\ell^4}{2^{2n}} + \frac{16\ell^2 q^2}{2^{2n}} . \tag{3.6}$$

Again, we look at the second term. We can fix the values to be arbitrary $s \neq s'$. Note that conditioning could impact the randomness of the function F. However, when we run on fixed inputs $\overline{x}, \overline{x}'$. This would mean that F is uniformly random conditioned on the set of queries in L and the set of evaluations of $\overline{x}, \overline{x}'$ resulting in states s, s' respectively. In this setting, define a bad

event as an event $\mathcal{E}$ such that the final output are not the same. Therefore, we can bound:

$$\Pr[\tilde{Y}_\ell = \tilde{Y}'_\ell] \leq \Pr[\tilde{Y}_\ell = \tilde{Y}'_\ell | \mathcal{E}] + \Pr[\neg\mathcal{E}] \leq \frac{1}{2^r} + \frac{q+\ell+1}{2^n}$$

where the last inequality is a result of Lemma 5.14.

We can look at $\Pr[\neg\mathcal{E}]$ as conditioned on an event that the output of $F(0, \text{opad})$ collides with one of the other at most $q+\ell$ values corresponding to the L and the evaluations of the inputs. When this output does not collide, then the final output is freshly sampled and by the randomness of F we get that the final outputs can collide with probability at most $\frac{1}{2^n}$. Furthermore, the probability that output of $F(0, \text{opad})$ collides is $\frac{q+\ell}{2^n}$. This concludes the proof.

□

3.5.5 CBC-Based Extractors Are Insecure

A natural candidate for an online seedless extractor is using a permutation in CBC mode. A CBC-based extractor construction uses a permutation $\pi : \{0,1\}^n \to \{0,1\}^n$ to absorb n-bit inputs. Its refresh function is defined as

$$\mathsf{refresh}^\pi(s, x) \;=\; \pi(s \oplus x)\ .$$

However, it turns out that this approach does not lead to a secure extractor. This section presents a simple attack against CBC-based extractors. The attack works irrespective of how the finalization function is defined.

Theorem 3.24 (Attack against CBC Extractors)**.** *Let* refresh *as defined above. There exists an ℓ-legitimate q-attacker $\mathcal{A}$ with black-box access to a function* finalize*, such that for all* CBC $=$ (refresh, finalize)

$$\mathrm{Adv}^{\mathrm{ext},\pi}_{\mathrm{CBC}}(\mathcal{A}) \;=\; 1 - 2^{-(r-1)}\ ,$$

where r is the output length of the extractor, $q = 2\ell + 2\alpha$, and α is the query complexity of finalize.

The idea of the attack is to have the attacker create the i^{th} input block as either $\pi^i(0^n) \oplus \pi^i(1^n)$ or 0, each with probability $1/2$.[9] After ℓ such steps, the attacker will have provided ℓ bits of entropy (even conditioned on its π-queries), but only a single bit will have accumulated in the state, which will be $\pi^i(0^n)$ or $\pi^i(1^n)$, each with probability $1/2$.

Proof. Consider a two-stage attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. $\mathcal{A}_1$ works as follows:

1. Initially, set $a = 0^n$ and $b = 1^n$.
2. For blocks $i = 1$ to ℓ:
   (a) Set $x_{i,0} = 0$ and $x_{i,1} = a \oplus b$. Choose random bit $\beta \leftarrow \{0, 1\}$ and set $x_i = x_{i,\beta}$.
   (b) Set $a \leftarrow \pi(a)$ and $b \leftarrow \pi(b)$.
3. Set state information to $\sigma \leftarrow (a, b)$. In particular, *forget* all values $x_{i,b}$ and all values of β. Return σ and $x = (x_1, \ldots, x_\ell)$.

$\mathcal{A}_2$, given $\sigma = (a, b)$ and y as input, proceeds as follows:

1. Compute finalize(a) and finalize(b). If either of them equals y, return 0; else, return 1.

Assume the initial state of the extractor is 0^n. In order to understand the attack, consider the state of the extractor after XORing the first input x_1: it is either 0^n or 1^n, each with probability $1/2$. After applying refresh, which consists of just applying π to the state, the new state is either $\pi(0^n)$ or $\pi(1^n)$, again with probability $1/2$ each.

The second input x_2 is either 0^n or $\pi(0^n) \oplus \pi(1^n)$; XORing it to the state results in either $\pi(0^n)$ or $\pi(1^n)$. After applying refresh, the new state is $\pi^2(0^n)$ or $\pi^2(1^n)$. Extending this argument to all ℓ inputs, the state of the extractor after absorbing them is either $a = \pi^\ell(0^n)$ or $a = \pi^\ell(1^n)$. After calling finalize, the state is either finalize(a) or finalize(b). Therefore, in the real world ($b = 0$),

[9] Here, π^i denotes the i-fold application of π.

$\mathcal{A}_2$ will always output 0, whereas in the ideal world ($b = 1$) this will only happen with probability at most $2^{-(r-1)}$.

In terms of efficiency, observe that $\mathcal{A}_1$ makes 2ℓ queries to π, and $\mathcal{A}_2$ makes twice the number α of π-queries required to evaluate a call to finalize.

Finally, in order to show that $\mathcal{A}$ is legitimate, observe first that the state σ can be computed from the queries made by $\mathcal{A}_1$. It easily seen that given only the queries, the vector $x = (x_1, \ldots, x_\ell)$ has ℓ bits of entropy. □

3.5.6 Seedless HKDF

This section heuristically argues about the standard-model security of the well-known HKDF key-derivation function (KDF). KDFs are used in practice to derive random-looking key material from high-entropy sources. For self-containment this section first presents the syntax of and a security definition for KDFs.

HKDF [Kra10] is a generic KDF construction based on an extractor and a pseudorandom function (PRF), where both the extractor and the PRF are instantiated with HMAC. The original HKDF construction is seeded since the extractor used is seeded. This section presents an unseeded version of HKDF by using a seedless extractor instead. HMAC is analyzed as extractor and PRF in the ideal model with a random compression function. By combining the standard-model versions of the extractor and the PRF (based, e.g., on the Davies-Meyer compression function), one obtains a KDF in the standard model.

Syntax. A *seedless key-derivation function (SL-KDF)* KDF is an algorithm that takes as input

- source material $x \in \{0,1\}^m$,
- context information c, as well as
- a desired output length o

and outputs an o-bit string

$$y \leftarrow \mathsf{KDF}(x, c, o) .$$

Security. The security definition for seedless KDFs—in the standard model—considers a two-stage attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. The first stage $\mathcal{A}_1$ outputs a value x and some state information σ for $\mathcal{A}_2$. The second stage attacker $\mathcal{A}_2$, based on σ, may make particular queries (described below) and outputs a single bit at the end. Formally, the game proceeds as follows:

1. The challenger chooses $b \leftarrow\!\!\$ \{0, 1\}$.
2. $\mathcal{A}_1$ produces $(\sigma, x) \leftarrow\!\!\$ \mathcal{A}_1$.
3. The second-stage attacker $\mathcal{A}_2$ is given σ and may make construction queries (c, o), which are answered by $\mathsf{KDF}(x_1, \ldots, x_\ell, c, o)$ if $b = 0$, and by a uniformly random value in $\{0, 1\}^o$ if $b = 1$. $\mathcal{A}_2$ is restricted to a single query for every value c.
4. At the end, $\mathcal{A}_2$ outputs a decision bit b' and wins the game if and only if $b' = b$.

The advantage of $\mathcal{A}$ in this extraction game is denoted by $\mathrm{Adv}^{\mathrm{kdf}}_{\mathsf{KDF}}(\mathcal{A})$.

An attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is called an $(\mathcal{F}, T, q)$-attacker if (1) the distribution of the values (σ, x) produced by $\mathcal{A}_1$ is from a class of distributions $\mathcal{F}$, (2) $\mathcal{A}_2$ runs in time at most T, and (3) makes at most q queries to the KDF. The above leads to the following definition of seedless key-derivation functions in the standard model:

Definition 3.25. An algorithm KDF is a $(\mathcal{F}, T, q, \varepsilon)$-*SL-KDF* if for every $(\mathcal{F}, T, q)$-attacker $\mathcal{A}$,

$$\mathrm{Adv}^{\mathrm{kdf}}_{\mathsf{KDF}}(\mathcal{A}) \leq \varepsilon .$$

Pseudorandom functions. A *variable-length pseudorandom function (VL-PRF)* is a function $\mathsf{PRF}(k, c, o)$ that, on a uniformly random key, is computationally indistinguishable from a truly

random function that produces outputs of the desired length. Specifically, the security of a VL-PRF can be captured in the P-model by giving an attacker $\mathcal{A}$ access to P and allowing it to make *construction* queries to either $\mathsf{PRF}(k, \cdot, \cdot)$ on a random key or to a truly random function that outputs a random and independent value from $\{0,1\}^o$ for every input (c, o); $\mathcal{A}$ is restricted to make at most one construction query for every value c. Function PRF is called a (p, q, ε)-*VL-PRF in the P-model* if an attacker making at most p queries to P and at most q construction queries has advantage at most ε of in telling PRF apart from a random function in the above experiment. Similarly, one can define a (T, q, ε)-VL-PRF in the standard model, where T is the running time of $\mathcal{A}$.

Extract-then-expand KDFs and HKDF. Seedless KDFs can be generically built by composing a seedless extractor ext with a variable-length PRF PRF as follows:

$$\mathsf{KDF}(x, c, o) \;:=\; \mathsf{PRF}(\mathsf{ext}(x), c, o)\ .$$

Observe that the definition of seedless extractor (Definition 3.7) can be modified to a standard-model definition of $(\mathcal{F}, T, \varepsilon)$-extractors (similarly to Definition 3.25):

Definition 3.26. An algorithm ext is a seedless $(\mathcal{F}, T, \varepsilon)$-*extractor* if for every $(\mathcal{F}, T)$-attacker $\mathcal{A}$,

$$\mathrm{Adv}^{\mathsf{ext}}_{\mathsf{ext}}(\mathcal{A}) \;\leq\; \varepsilon\ .$$

The following theorem by [Kra10] is easy to prove (for most reasonable function classes $\mathcal{F}$).

Theorem 3.27. *Let* ext *be a seedless $(\mathcal{F}, T, \varepsilon)$-extractor and let* PRF *be a (T', q', ε')-VL-PRF. Then,* KDF *as defined above is a seedless $(\mathcal{F}, \min(T, T'), q, \varepsilon + \varepsilon')$-KDF.*

Recall that HKDF instantiates both the extractor and the VL-PRF with HMAC (cf. Section 3.5.4). The security of HKDF as a KDF can be argued heuristically in the standard model as follows: First,

one infers the standard-model security of HMAC as an extractor—from Theorem 3.20. In particular, the heuristic assumption is that the extractor security of HMAC in the random-compression-function model implies security of HMAC in the standard model against all distributions from the class $\mathcal{F}_{\text{practice}}$ of sources encountered in practice. Second, one infers the standard-model security of HMAC as a VL-PRF—from the well-known fact that HMAC is a good VL-PRF in the random-compression-function model.

If the composed construction is not a secure standard-model KDF, then this would constitute a natural counterexample to the random-oracle methodology: by virtue of Theorem 3.27, either HMAC is not a good standard-model extractor (but secure in the idealized model), or HMAC is not a good standard-model VL-PRF (but secure in the idealized model). Such a non-contrived counterexample would be a remarkable breakthrough in the area of ideal-model security proofs.

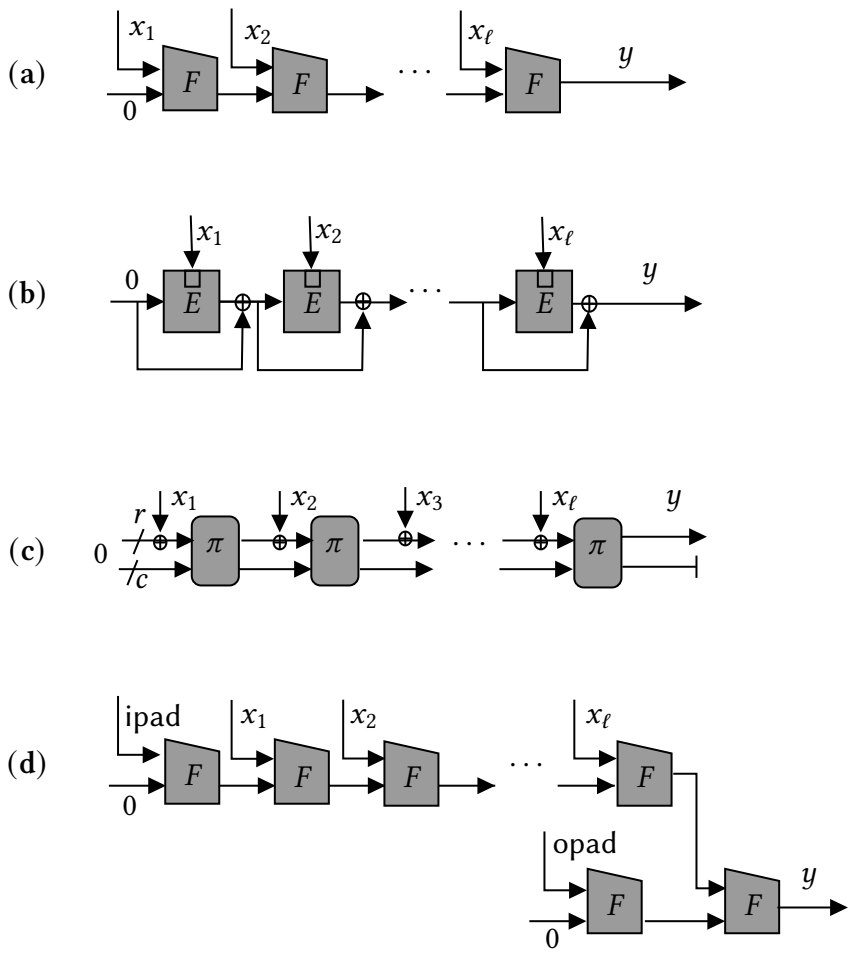

Figure 3.1: Four online computational extractors are represented in this diagram. These are based on: **(a)** Merkle-Damgård with a random compression function; **(b)** Merkle-Damgård with Davies-Meyer; **(c)** Sponge; and **(d)** HMAC.
Each extractor is shown to process inputs $x_1 \ldots x_\ell$ (calls to refresh) to compute the output y (call to finalize). The IT variant truncates the output y and returning the first r bits. However, note that the Sponge construction needs a truncated output even for the computational variant.

4 | Seedless Pseudorandom Number Generation

This chapter is based on joint work with Sandro Coretti, Yevgeniy Dodis, and Stefano Tessaro that appeared in CRYPTO 2019 [CDKT19a]. Some passages are taken verbatim from the full version of this paper [CDKT19b].

4.1 Pseudorandom Number Generators with Input

A *pseudorandom number generator with input (PRNG)* is a stateful cryptographic primitive. It gradually accumulates entropy in its state by absorbing inputs and can be used to output pseudorandom bits once the entropy of the state is sufficiently high. Moreover, it is both forward and backward secure, i.e., past outputs remain random upon future state compromise, and, by absorbing sufficient amounts of entropy, a PRNG can recover from state compromise.

This section introduces a novel security definition for PRNGs that differs from previous notions in several crucial ways. Specifically, a comparison to the original *robustness* notion by Dodis *et al.*[DPR+13], based on work by Barak and Halevi [BH05], as well as to an adaptation of it by Gazi and Tessaro [GT16] for idealized models is provided in Appendix 4.4.

Much like our two notions of security for extractors, in this section we define computational PRNGs and information-theoretically secure (IT) PRNGs. IT PRNGs differ from computational

PRNGs in that once the attacker stops interacting with the PRNG, the output of the PRNG remains random even if the attacker is given the entire function table of the underlying ideal primitive. That is, IT PRNGs achieve so-called *everlasting security* (cf. works in the hybrid bounded-storage model by Harnik and Naor [HN06]). This distinction is analogous to that between seedless extractors and IT seedless extractors (cf. Section 3).

4.1.1 Syntax

A PRNG consists of two algorithms: one for absorbing new inputs and one for producing pseudorandom outputs. Formally, it is defined as follows:

Definition 4.1 (Syntax of PRNGs)**.** A *pseudorandom number generator with input (PRNG)* is a pair of algorithms $\mathsf{PRNG} = (\mathsf{refresh}, \mathsf{next})$ having access to an ideal primitive P and sharing an n-bit state s, where

- $\mathsf{refresh}$ takes a state s and an input $x \in \{0,1\}^m$ and produces a new state $s' \leftarrow \mathsf{refresh}^P(s, x)$, and
- next takes a state s and produces a new state and an output $y \in \{0,1\}^r$, i.e, $(s', y) \leftarrow \mathsf{next}^P(s)$.

A PRNG processing m-bit inputs and producing r-bit output is called a (m, r)*-PRNG*.

4.1.2 Security Game

Robustness game. PRNGs are expected to satisfy the so-called *robustness* property, which captures the properties discussed at the beginning of Section 4.1. The corresponding security game is depicted in Figure 4.1. The game initially chooses a random bit b and initializes the state of the PRNG to 0^n. Subsequently, it offers the following oracles to $\mathcal{A}$:

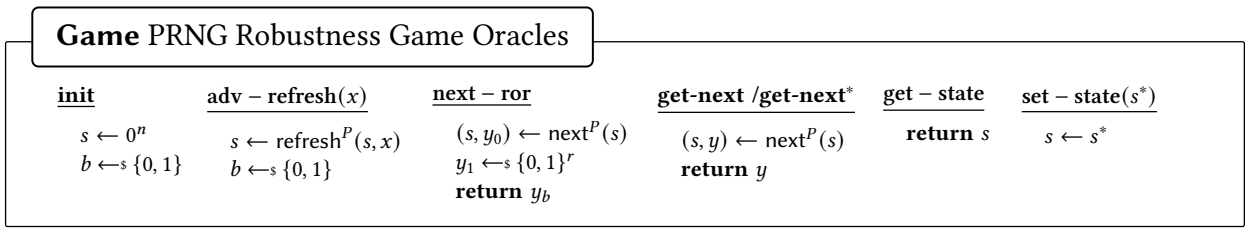

Figure 4.1: Oracles for the PRNG Robustness Game

- **adv − refresh**(x) calls the refresh procedure to absorb $x \in \{0,1\}^n$ into the internal state of the PRNG;
- **get-next** and **get-next***allow the attacker to get pseudorandom outputs by calling the next procedure on the current state and returning the output y. The difference between the two oracles is that **get-next** is supposed to be called only when the state has high entropy, whereas **get-next***can be called *prematurely*, i.e., before the state has absorbed enough randomness for the next function to output pseudorandom values (cf. definition of legitimate attackers below).
- **next-ror** works like the **get-next** -oracles, except that it creates a challenge, i.e., if $b = 1$, it outputs a uniform random value $y_1 \in \{0,1\}^r$ instead of the PRNG output y_0.
- **get-state** and **set-state**model state compromises by letting the attacker learn the current state or set it to an arbitrary value, respectively.

The advantage of $\mathcal{A}$ in the robustness game is denoted by $\text{Adv}^{\text{rob},P}_{\text{PRNG}}(\mathcal{A})$.

Canonical attackers. It will be useful to define to following notion of canonical attackers: Consider the interaction of an attacker $\mathcal{A}$ with the robustness game. The following events are called *entropy drains*:

- the beginning of the game,
- calls to **get-state** or **set-state**, and

- calls to **get-next***.

In other words, entropy drains are the events that cause the PRNG state to lose its entropy, which includes premature calls to next. An attacker $\mathcal{A}$ is said to be *canonical* if it does not make **get-next*** queries nor the following query pattern: an entropy drain followed by one or more **adv-refresh** queries, followed by a **get-state** query.

Considering canonical attackers only is without loss of generality. This is because the above sequence of queries can be simulated by the attacker by making a **get-state** query right away and computing the output of **get-state** or **get-next*** itself. In particular, for every attacker $\mathcal{A}$, there exists a canonical attacker $\mathcal{A}$ with the same advantage. All attackers in the remainder of this work are therefore assumed to be canonical.

Legitimate attackers. In order to obtain a sensible definition devoid of trivial attacks, attackers must satisfy a "legitimacy" condition. The condition roughly requires that an attacker only ask for challenges when it has sufficient amount of uncertainty about the PRNG's internal state.

Towards formalizing the legitimacy condition, consider the interaction of $\mathcal{A}$ with a *variant* of the robustness game defined as follows: Whenever oracles **next-ror** or **get-next** are called, instead of evaluating next, the game simply uses two uniformly random and independent values (s, y) as the output of next.

Observe that this variant of the robustness game, called the *legitimacy game* corresponds to an interaction between $\mathcal{A}$ and an *ideal* PRNG, which produces perfect randomness. Moreover, the legitimacy game is *construction-independent.*

In the legitimacy game, define now the following random variables immediately before $\mathcal{A}$ makes the i^{th} call to oracle **get-next** or **next-ror** :

- $\mathcal{L}_i$: the list of P-queries by $\mathcal{A}$ and the corresponding answers;
- Σ_i: the state of $\mathcal{A}$;

- $\overline{X}_i$: vector of inputs provided by $\mathcal{A}$ since the *the most recent entropy drain (MRED)*; and
- S_i: the state of the PRNG immediately after the MRED.

The legitimacy condition requires that $\mathcal{A}$ provide inputs that have high min-entropy even conditioned on its current state, the queries so far, and the state of the PRNG after the MRED.

Definition 4.2 (Legitimate attackers)**.** An attacker $\mathcal{A}$ is said to be γ^**-legitimate* if for all i,

$$\mathrm{H}_\infty(\overline{X}_i|\Sigma_i\mathcal{L}_iS_i) \;\geq\; \gamma^*\,,$$

where MREDs are defined as above.

In order to capture IT-legitimate attackers (against IT PRNGS), the set of entropy drains is extended to include

- calls to **get-next** and **next-ror** .

With this definition of MRED and notation analogous to that in the previous definition, IT-legitimate attackers are defined as follows:

Definition 4.3 (Legitimate IT attackers)**.** An attacker $\mathcal{A}$ is said to be γ^**-IT-legitimate* if for all i,

$$\mathrm{H}_\infty(\overline{X}_i|\Sigma_i\mathcal{L}_iS_i) \;\geq\; \gamma^*\,,$$

w.r.t. the extended definition of MRED.

Robust PRNGs. We are now ready to quantify the efficiency of attacker $\mathcal{A}$, and to define our final notion of PRNG robustness.

Definition 4.4 (Attacker efficiency)**.** An attacker is called a (q, t, ℓ)*-attacker* if

- q is the maximum number of P-queries it makes,

- ℓ is the maximum number of **adv-refresh**calls between any entropy drain and successive call to either **next-ror** or **get-next** , and
- t is the maximum total number of calls to any oracle in the robustness game other than **adv-refresh**.

An attacker is called a (q, t, ℓ)-*IT-attacker* if it satisfies the above conditions but makes an arbitrary number of queries to P *after* the interaction with the challenger ends.

Definition 4.5 (Robustness of PRNGs)**.** A PRNG construction $\mathsf{PRNG} = (\mathsf{refresh}, \mathsf{next})$ with oracle access an ideal primitive P is $(\gamma^*, q, t, \ell, \varepsilon)$-*(IT-)robust* in the P-model if for every γ^*-(IT-)legitimate (q, t, ℓ)-(IT-)attacker,

$$\mathrm{Adv}^{\mathrm{rob},P}_{\mathsf{PRNG}}(\mathcal{A}) \;\leq\; \varepsilon\,.$$

Observe that online extractors (cf. Definition 3.12) are a special case of robust PRNGs. In terms of construction, the PRNG next algorithm can be replaced by finalize, which simply discards the state output by next. If then the PRNG robustness game is relaxed such that the only queries the attacker can make are (a) arbitrarily many queries to **adv-refresh** followed by (b) $t = 1$ query to **next-ror** , one obtains a notion equivalent to Definiton 3.7.

4.2 Intermediate Computational PRNG Security Notions

In keeping with tradition in PRNG literature, it is useful to define two simple properties called *recovering* and *preserving*. Recovering security requires that that if after an entropy drain, sufficient amount of entropy has been absorbed into the PRNG state, the output of the next function look random. Preserving security asks that after absorbing adversarially chosen inputs, a high-entropy state not become compromised, and the output of the next function after the absorption look random.

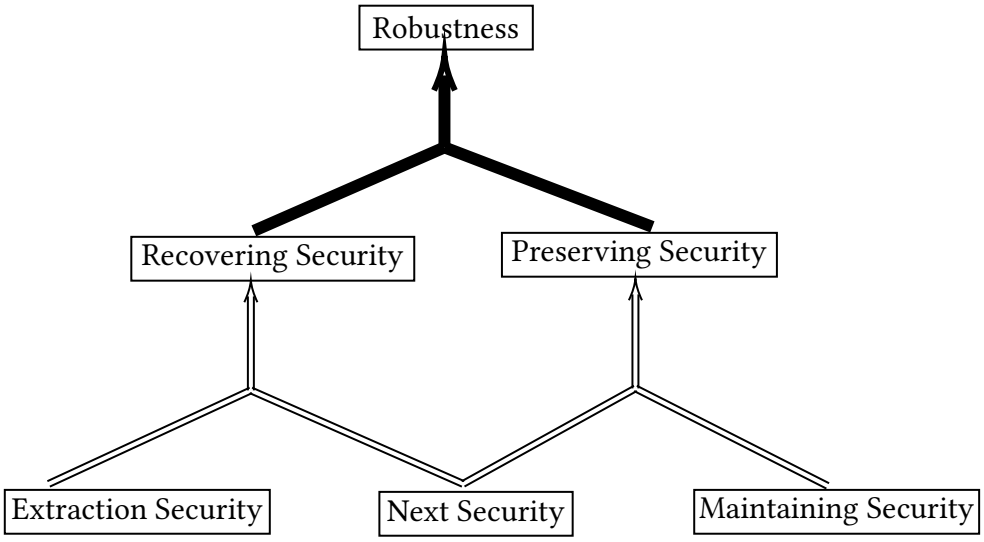

Figure 4.2: This figure represents the implication relations between the different intermediate notions of security. The filled arrows stand for a generic proof, while the unfilled arrows represent a construction-specific proof.

Recovering and preserving security can be shown to generically (i.e., for any PRNG construction) imply robustness. In order to establish these two properties themselves, it helps to introduce three further properties called *extraction*, *maintaining*, and *next security*:

- **Extraction security:** show that the PRNG *state* is indistinguishable from a uniform one after sufficient amounts of entropy have been absorbed;
- **Maintaining security:** show that the PRNG *state* is indistinguishable from a uniform one if it is random initially and arbitrary inputs are absorbed;
- **Next security:** show that the output of the next function is indistinguishable from a random value if it is called on a random input.

For each PRNG construction considered in this work, extraction and next security imply recovering security, and maintaining and next security imply preserving security. These proofs, however,

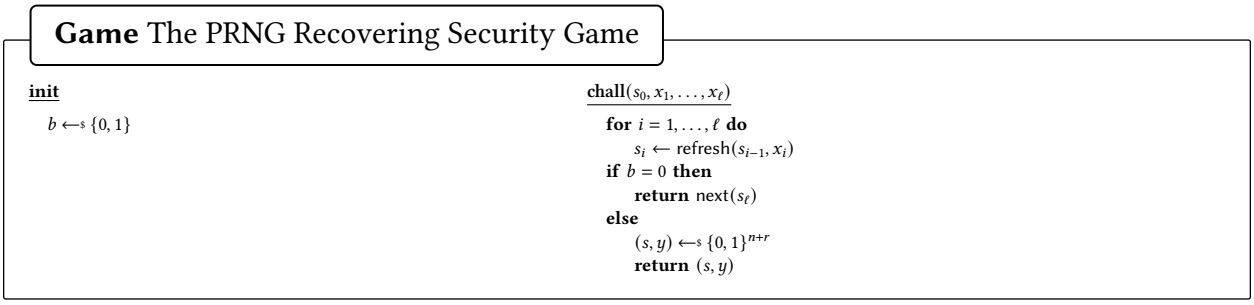

Figure 4.3: Oracles for PRNG Recovering Security game.

are *not* generic and must be repeated for each PRNG construction. Figure 4.2 illustrates these implications.

4.2.1 Recovering and Preserving Security

As stated above, it is useful to define two simple properties called *recovering* and *preserving*, which together generically imply robustness via a hybrid argument.

Recovering security. The intuition behind Recovering security is that if after an entropy drain, sufficient amount of entropy (from the perspective of the attacker) has been absorbed into the state, then the output of next is indistinguishable from a uniformly random value in $\{0,1\}^{n+r}$.

The corresponding game is depicted in Figure 4.3. It lets the attacker specify an initial state s_0 and a vector of inputs $x_1, \ldots, x_\ell$; the inputs are then absorbed one-by-one, and next is called on the resulting state. The game returns the output of next if $b = 0$ and a uniform value if $b = 1$. The advantage of an attacker $\mathcal{A}$ in the recovering game is denoted by $\mathrm{Adv}^{\mathrm{rec},P}_{\mathsf{PRNG}}(\mathcal{A})$.

Similarly to the robustness game, an attacker has to satisfy a legitimacy condition. In particular, $\mathcal{A}$ is γ^*-legitimate if

$$\mathrm{H}_\infty(X_1, \ldots, X_\ell | \Sigma \mathcal{L} S_0) \;\geq\; \gamma^* \,,$$

where

- Σ is the state of $\mathcal{A}$ just before the call to **chall**,
- $\mathcal{L}$ is the list of query and answers $\mathcal{A}$ has made to P up to the call to **chall**, and
- S_0 is the initial state that the adversary provides.

For the recovering game, q again denotes the maximum number of P-queries that $\mathcal{A}$ makes, and ℓ is the maximum number of blocks with which it calls oracle **chall**; a corresponding attacker is referred to as (q, ℓ)-*attacker*.

Definition 4.6. A PRNG construction PRNG = (refresh, next) is said to be $(\gamma^*, q, \ell, \varepsilon)$-*recovering* in the P-model if for every γ^*-legitimate (q, ℓ)-attacker,

$$\mathrm{Adv}^{\mathrm{rec},P}_{\mathrm{PRNG}}(\mathcal{A}) \;\leq\; \varepsilon\,.$$

Preserving security. At a high level, preserving security requires that by absorbing adversarially chosen inputs, a high-entropy state cannot become compromised, and the output of next after the absorption is indistinguishable from a uniformly random value in $\{0,1\}^{n+r}$.

The corresponding game is depicted in Figure 4.4. It lets the attacker specify a vector of inputs $x_1, \ldots, x_\ell$; the inputs are then absorbed into a *randomly chosen* state one-by-one, and next is called on the resulting state. The game returns the output of next if $b = 0$ and a uniform value if $b = 1$. The advantage of an attacker $\mathcal{A}$ in the preserving game is denoted by $\mathrm{Adv}^{\mathrm{pre},P}_{\mathrm{PRNG}}(\mathcal{A})$. There is no legitimacy constraint on $\mathcal{A}$; the parameters q and ℓ are defined as before for a (q, ℓ)-attacker.

Definition 4.7. A PRNG construction PRNG = (refresh, next) is said to be (q, ℓ, ε)-*preserving* in the P-model if for every (q, ℓ)-attacker,

$$\mathrm{Adv}^{\mathrm{pre},P}_{\mathrm{PRNG}}(\mathcal{A}) \;\leq\; \varepsilon\,.$$

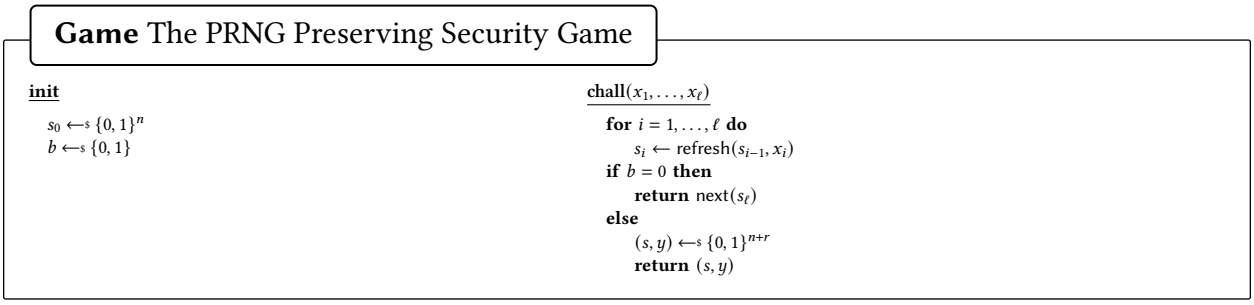

Figure 4.4: Oracles for PRNG Preserving Security game.

Recovering and preserving imply robustness. As mentioned above, in order to establish robustness of a PRNG construction, it suffices to prove that it is both *recovering* and *preserving*.

Theorem 4.8. *Consider a PRNG construction* $\mathsf{PRNG} = (\mathsf{refresh}, \mathsf{next})$ *for which* refresh *makes* α *P-calls and* next *makes* β *P-calls. Furthermore, assume* PRNG *is both*

- $(\gamma^*, q, \ell, \varepsilon_{\mathrm{rec}})$*-recovering and*
- $(q, \ell, \varepsilon_{\mathrm{pre}})$*-preserving*

in the P-model. Then, PRNG *is also* $(\gamma^*, q, t, \ell, \varepsilon_{\mathrm{rob}})$*-robust in the P-model, where*

$$\varepsilon_{\mathrm{rob}} \;\leq\; t \cdot (\varepsilon_{\mathrm{rec}} + \varepsilon_{\mathrm{pre}}) \;.$$

For the proof of Theorem 4.8, consider a γ^*-legitimate (canonical) (q, t, ℓ)-attacker $\mathcal{A}$. The proof is through a series of hybrid games, where the advantage of $\mathcal{A}$ in the final game is easily seen to be 0. In order to define the hybrids, let a *nice* next query be either **get-next** or **next-ror** . A nice next query is • *recovering* if it is the first such query after the MRED and • *preserving* otherwise. Define the hybrid rob_i to be the robustness game for PRNG where for first i nice next queries, the output of next is replaced by a uniform random string of length $n + r$. Moreover, consider an intermediate hybrid $\mathrm{rob}_{i+\frac{1}{2}}$, where the challenger also replaces the output of next with a random string if the $(i+1)^{\mathrm{st}}$ query is *preserving*.

Denote by η_i the probability that $\mathcal{A}$ outputs 1 when interacting with hybrid rob_i. In the following, preserving security will be used to argue the indistinguishability of hybrids rob_i and $\text{rob}_{i+\frac{1}{2}}$ (Claim 4.9), and recovering security (Claim 4.10) for $\text{rob}_{i+\frac{1}{2}}$ and rob_i.

Claim 4.9. For all $i = 0, \ldots, t$, $|\eta_i - \eta_{i+\frac{1}{2}}| \leq \varepsilon_{\text{pre}}$.

Proof. The two hybrids only differ in the case when $(i+1)^{\text{st}}$ next query is *preserving*. Hence, assume that the adversary $\mathcal{A}$ ensures that the $(i+1)^{\text{st}}$ query is indeed preserving, which serves to maximize its advantage. Consider the following attacker $\mathcal{A}'$ against the preserving security of PRNG: Initially, $\mathcal{A}'$ sets

- $b \leftarrow_{\$} \{0,1\}$,
- $s \leftarrow 0^n$,
- $j \leftarrow 0$, and
- $\chi \leftarrow \lambda$, where λ is the empty string.

Then, $\mathcal{A}'$ runs $\mathcal{A}$, simulating all oracle calls made by $\mathcal{A}$ answering the queries from $\mathcal{A}$ as follows: At all times, P-queries by $\mathcal{A}$ are simply forwarded by $\mathcal{A}'$ to its own P-oracle and back. Furthermore, while $j \leq i$:

- **adv − refresh**(x): $\mathcal{A}'$ simply ignores the query.
- **set − state**(s'): $\mathcal{A}'$ just sets $s \leftarrow s'$.
- **get − state**: $\mathcal{A}'$ returns s.
- **get − next** or **next − ror**: $\mathcal{A}'$ chooses $(s, y) \leftarrow_{\$} \{0,1\}^{(n+r)}$ uniformly at random, increments $j \leftarrow j + 1$, and returns y.

Once $j = i + 1$, $\mathcal{A}'$ simulates the oracles as follows:

- **adv − refresh**(x): $\mathcal{A}'$ appends x to χ, i.e., $\chi \leftarrow \chi || x$.

- **set − state**(s'): $\mathcal{A}'$ sets $s \leftarrow s'$.
- **get − state**: $\mathcal{A}'$ returns s.
- **get − next**: $\mathcal{A}'$ calls its challenge oracle to obtain $(s, y) \leftarrow$ **chall**(s, χ), increments $j \leftarrow j+1$, and returns y.
- **next − ror**: $\mathcal{A}'$ calls its challenge oracle to obtain $(s, y_0) \leftarrow$ **chall**(s, χ), increments $j \leftarrow j+1$. It then chooses $y_1 \leftarrow\!\!\$\ \{0,1\}^r$ and returns y_b.

Subsequently, i.e., once $j > i+1$, $\mathcal{A}'$ uses the state s returned by **chall** and its P-access to keep simulating the oracles consistent with rob_{i+1}. In the end, $\mathcal{A}'$ outputs whatever $\mathcal{A}$ outputs.

Let the challenge bit of the challenger for $\mathcal{A}'$ be $\tilde{b}$. Note that when $\tilde{b} = 0$, $\mathcal{A}'$ perfectly simulates hybrid $\mathrm{rob}_{i+\frac{1}{2}}$ for $\mathcal{A}$. Similarly, if $\tilde{b} = 1$, $\mathcal{A}'$ perfectly simulates hybrid rob_{i+1} for $\mathcal{A}$. (Recall that $\mathcal{A}$ is canonical.) □

The next step is to show that the hybrids $\mathrm{rob}_{i+\frac{1}{2}}$ and rob_{i+1} are indistinguishable from each other.

Claim 4.10. For all $i = 0, \ldots, t-1$, $|\eta_{i+\frac{1}{2}} - \eta_{i+1}| \leq \varepsilon_{\mathrm{rec}}$.

Proof. The two hybrids only differ in the case when $(i+1)^{\mathrm{st}}$ next query is *recovering*. Hence, assume that the adversary $\mathcal{A}$ ensures that the $(i+1)^{\mathrm{st}}$ query is indeed recovering, which serves to maximize its advantage. Consider the following attacker $\mathcal{A}'$ against the recovering security of PRNG: Initially, $\mathcal{A}'$ sets

- $b \leftarrow\!\!\$\ \{0,1\}$,
- $s \leftarrow 0^n$,
- $j \leftarrow 0$, and
- $\chi \leftarrow \lambda$, where λ is the empty string.

Then, $\mathcal{A}'$ runs $\mathcal{A}$, simulating all oracle calls made by $\mathcal{A}$ answering the queries from $\mathcal{A}$ as follows: At all times, P-queries by $\mathcal{A}$ are simply forwarded by $\mathcal{A}'$ to its own P-oracle and back. Furthermore, while $j \leq i$:

- **adv − refresh**(x): $\mathcal{A}'$ simply ignores the query.
- **set − state**(s'): $\mathcal{A}'$ just sets $s \leftarrow s'$.
- **get − state**: $\mathcal{A}'$ returns s.
- **get − next** or **next − ror**: $\mathcal{A}'$ chooses $(s, y) \leftarrow\!\!\$\ \{0,1\}^{(n+r)}$ uniformly at random, increments $j \leftarrow j+1$, and returns y.

Once $j = i+1$, $\mathcal{A}'$ simulates the oracles as follows:

- **adv − refresh**(x): $\mathcal{A}'$ appends x to χ, i.e., $\chi \leftarrow \chi||x$.
- **set − state**(s'): $\mathcal{A}'$ sets $s \leftarrow s'$.
- **get − state**: $\mathcal{A}'$ returns s.
- **get − next**: $\mathcal{A}'$ calls its challenge oracle to obtain $(s, y) \leftarrow$ **chall**(s, χ), increments $j \leftarrow j+1$, and returns y.
- **next − ror**: $\mathcal{A}'$ calls its challenge oracle to obtain $(s, y_0) \leftarrow$ **chall**(s, χ), increments $j \leftarrow j+1$. It then chooses $y_1 \leftarrow\!\!\$\ \{0,1\}^r$ and returns y_b.

Subsequently, i.e., once $j > i+1$, $\mathcal{A}'$ uses the state s returned by **chall** and its P-access to keep simulating the oracles consistent with rob_{i+1}. In the end, $\mathcal{A}'$ outputs whatever $\mathcal{A}$ outputs.

Let the challenge bit of the challenger for $\mathcal{A}'$ be $\tilde{b}$. Note that when $\tilde{b} = 0$, $\mathcal{A}'$ perfectly simulates hybrid $\text{rob}_{i+\frac{1}{2}}$ for $\mathcal{A}$. Similarly, if $\tilde{b} = 1$, $\mathcal{A}'$ perfectly simulates hybrid rob_{i+1} for $\mathcal{A}$. (Recall that $\mathcal{A}$ is canonical.)

It remains to argue that $\mathcal{A}'$ is γ^*-legitimate. To that end, recall from Section 4.1.2 that the legitimacy condition is defined in the *legitimacy* game. Observe in particular, that up to the time $\mathcal{A}'$ makes its challenge queries, $\mathcal{A}$'s view is exactly as it would be in the legitimacy game, in which one considers the following random variables immediately before $\mathcal{A}$ makes the i^{th} call to oracle **get-next** or **next-ror** :

- $\mathcal{L}_i$: the list of P-queries by $\mathcal{A}$ and the corresponding answers;
- Σ_i: the state of $\mathcal{A}$;
- $\overline{X}_i$: vector of inputs provided by $\mathcal{A}$ since the *the most recent entropy drain (MRED)*; and
- S_i: the state of the PRNG immediately after the MRED.

In a similar fashion, consider the following random variables pertaining to $\mathcal{A}'$ just before its call to the challenge oracle:

- X, the input vector $\mathcal{A}'$ provides to **chall**;
- Σ, the state of $\mathcal{A}'$ just before the call to **chall**;
- $\mathcal{L}$, the list of P-queries and answers by $\mathcal{A}'$ before the call to **chall**; and
- S_0; the state $\mathcal{A}'$ provides to **chall**.

That is, it needs to be established that

$$\mathrm{H}_\infty(X|\Sigma\mathcal{L}S_0) \;\geq\; \gamma^* \,. \tag{4.1}$$

using the argument of the γ^*-legitimacy of $\mathcal{A}$,

$$\mathrm{H}_\infty(\overline{X}_i|\Sigma_i\mathcal{L}_iS_i) \;\geq\; \gamma^* \,.$$

First, it is easily verified that $S = S_i$. Second, observe that Σ only needs to contain, in addition to Σ_i, the values of b and j, which are clearly independent of X. Third, clearly $\mathcal{L} = \mathcal{L}_i$. Finally, note that $X = \overline{X}_i$. From the preceding, (4.1) follows. □

4.2.2 Extraction, Maintaining, and Next Security

Extraction security. Extraction security requires that after absorbing blocks with high joint entropy, the state of the PRNG be indistinguishable from a uniformly random one. The corresponding game is a variant of the game for recovering security (cf. Figure 4.3) in which next is not applied to s_ℓ; instead, s_ℓ or a uniformly random value is output, depending on whether $b = 0$ or $b = 1$.

The legitimacy of an attacker $\mathcal{A}$ as well as parameters q and ℓ are defined identically to the recovering game (cf. Section 4.1.2); the advantage of $\mathcal{A}$ against extraction security of PRNG in the P-model is denoted by $\mathrm{Adv}^{\mathrm{ext},P}_{\mathsf{PRNG}}(\mathcal{A})$.

Maintaining security. Maintaining security is a variant of the preserving game (cf. Figure 4.3) in which next is not applied to s_ℓ; instead, s_ℓ or a uniformly random value is output, depending on whether $b = 0$ or $b = 1$. The parameters q and ℓ are defined identically to the preserving game (cf. Section 4.1.2); the advantage of $\mathcal{A}$ against maintaining security of PRNG in the P-model is denoted by $\mathrm{Adv}^{\mathrm{mtn},P}_{\mathsf{PRNG}}(\mathcal{A})$.

Next security. Next security requires that the output of the next function on a uniformly random state be indistinguishable from a uniformly random string. That is, an attacker $\mathcal{A}$, making at most q queries to the ideal primitive P, tries to distinguish $\mathsf{next}^P(S)$ from U_{n+r} for a uniformly random $S \in \{0,1\}^n$. Denote by $\mathrm{Adv}^{\mathrm{next},P}_{\mathsf{PRNG}}(\mathcal{A})$ the advantage of $\mathcal{A}$.

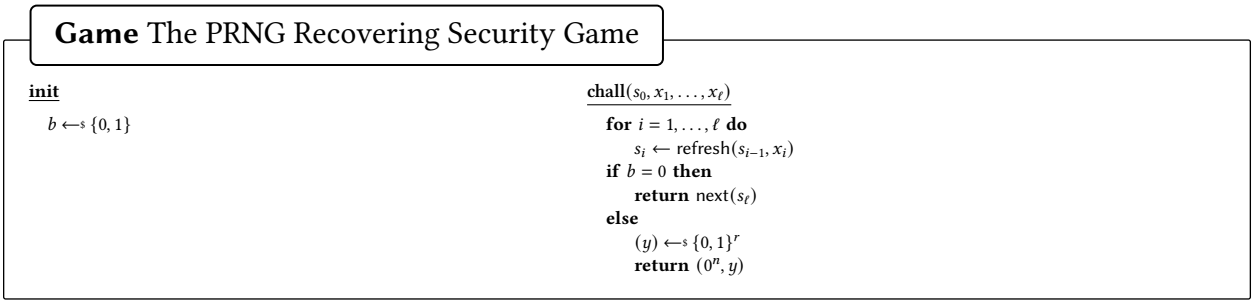

Figure 4.5: Oracles for PRNG Recovering Security game.

4.3 Intermediate IT PRNG Security Notions

Similarly to the computational case, it is useful to consider a simplified security notion, *recovering* security, for IT PRNGs. Recall that for the computational case, recovering security requires that the output of the next function next look random once sufficient amounts of entropy have been accumulated in the PRNG's state. In the IT case, the requirement is relaxed by only requiring that the output be indistinguishable from $(0^n, U_r)$. That is, call to next always resets the PRNG state to 0^n. This is without loss, as the definition of legitimacy considers every call to next an entropy drain.

The IT recovering game is represented in Figure 4.5. The advantage of an attacker $\mathcal{A}$ in this game is denoted by $\mathrm{Adv}^{\text{rec-IT},P}_{\mathrm{PRNG}}(\mathcal{A})$. An attacker $\mathcal{A}$ in this game is γ^*-legitimate if

$$\mathrm{H}_\infty(X_1, \ldots, X_\ell | \Sigma \mathcal{L} S_0) \;\geq\; \gamma^* \,,$$

where

- Σ is the state of $\mathcal{A}$ just before the call to **chall**,
- $\mathcal{L}$ is the list of query and answers $\mathcal{A}$ has made to P up to the call to **chall**, and
- S_0 is the initial state that the adversary provides.

A (q, ℓ)-IT-attacker here is one that can make at most q queries to P before its challenge (and arbitrarily many afterwards), and such that the input to **chall** consists of at most ℓ blocks.

Definition 4.11. A PRNG construction $\mathsf{PRNG} = (\mathsf{refresh}, \mathsf{next})$ is said to be $(\gamma^*, q, \ell, \varepsilon)$-*IT-recovering* in the P-model if for every γ^*-IT-legitimate (q, ℓ)-IT-attacker,

$$\mathrm{Adv}^{\text{rec-IT},P}_{\mathsf{PRNG}}(\mathcal{A}) \;\leq\; \varepsilon \;.$$

Observe that the definition of recovering security for IT PRNGs is—up to syntactical differences—equivalent to the security of the PRNG as an extractor, which is obtained by replacing the PRNG next algorithm by an algorithm finalize that simply discards the state output by next. In other words, an IT-robust PRNG can be viewed as an IT-secure online extractor. In particular, the following theorem is in principle little more than just a union bound.

Theorem 4.12. *Let* $\mathsf{PRNG} = (\mathsf{refresh}, \mathsf{next})$ *be a PRNG for which* refresh *makes* α P*-calls and* next *makes* β P*-calls. Let the* PRNG *be* $(\gamma^*, q, \ell, \varepsilon_{\text{rec}})$*-IT-recovering in the* P*-model. Then,* PRNG *is also* $(\gamma^*, q, t, \ell, \varepsilon_{\text{rob}})$*-IT-robust in the* P*-model, where*

$$\varepsilon_{\text{rob}} \;\leq\; t \cdot \varepsilon_{\text{rec}} \;.$$

As seen in the proof of Theorem 4.8, define the hybrid rob_i to be the IT-robustness game for PRNG where for first i nice next queries, the output of next is replaced by a uniform random string of length $n + r$. We denote by η_i the probability that $\mathcal{A}$ outputs 1 when interacting with hybrid rob_i.

Claim 4.13. For all $i = 0, \ldots, t-1$, $|\eta_i - \eta_{i+1}| \leq \varepsilon_{\text{rec}}$.

The proof of Claim 4.13 is completely analogous to that of Claim 4.10 and is therefore omitted.

4.4 Comparison to Previous PRNG Security Notions

Overview. The first robustness notion for PRNGs was proposed by Barak and Halevi [BH05] and required that PRNGs be:

- *resilient*, i.e., the attacker cannot predict the output of the PRNG even if he can influence the inputs;
- *forward secure*, i.e., upon state compromise, previous outputs of the PRNG remain random; and
- *post-compromise secure*, i.e., if sufficient entropy is absorbed into the PRNG state, the outputs regain security.

The definition was developed further in a crucial way by Dodis *et al.*[DPR+13], who considered the setting where entropy only *gradually* accumulates in the state (as opposed to the model in [BH05], where inputs are required to have high entropy in order to recover from compromise. In order to analyse constructions such as sponges, Gazi and Tessaro [GT16] extended the robustness definition further to a setting with a public random permutation.

Distribution samplers. In order to capture distributions for which the PRNG is expected to be robust, Dodis *et al.*[DPR+13] considered distribution samplers Sam, which with every input x_i for the PRNG would also provide an entropy estimate γ_i and a leakage value z_i. A sampler was considered *legitimate* if every input x_i has the promised min entropy γ_i even conditioned on all *other* inputs (i.e., past and future) as well as on *all* values γ_j and z_j. A very important consequence of this definition is that (it can be shown that) extraction with such a sampler Sam is impossible without a randomly chosen seed about which the sampler has no information. Dodis *et al.*provided a construction that uses such a seed and whose refresh function actually is information-theoretically secure. However, security completely breaks down as soon as the sampler can depend on the seed,

which is why said construction was deemed unsuitable by practitioners since seed-independence of the input may not be guaranteed in practice.

In an effort to analyze more practical constructions Gazi and Tessaro [GT16] extended the robustness notion to incorporate ideal primitives. In particular, for their analysis of Sponge, they considered samplers Sam^{π} with oracle access to a random permutation π and required an entropy notion related to that above, but required, informally, that inputs be unpredictable even if in addition to the above, one also conditions on the queries by the sampler (albeit the sampler is allowed *private* queries for every input it generates). While the extension to ideal models allows to analyze constructions used in practice, the work by Gazi and Tessaro does not solve the need for a seed (about which Sam^{π} has no information), without which the aforementioned impossibility still holds.

Main differences between existing notions and the new one. The new definition put forth in this work differs from previous ones in several ways: First, the legitimacy notion is such that there is no need for a seed. Second, because of that, the sampler and the distinguisher are merged into a single attacker, which would not be possible with the previous notions due to the need to hide the seed from Sam. Third, the new legitimacy notion measures not the sum of entropies of particular blocks conditioned on each other, but the entropy of an entire sequence of inputs *as a whole*. Finally, the new notion dispenses with the need for entropy estimates by simply only considering attackers that provide inputs for which the average min-entropy at certain points is high enough conditioned on the state and the queries of the attacker. Observe that the entropy estimates were originally introduced to circumvent the impossibility of randomness estimation, which would have been needed in reductions to computationally secure primitives (such as using a PRG for the next function). Hence, entropy estimates were an unnatural artifact of the definition anyway.

5 | Constructions of Seedless PRNGs

This chapter is based on joint work with Sandro Coretti, Yevgeniy Dodis, and Stefano Tessaro that appeared in CRYPTO 2019 [CDKT19a]. Some passages are taken verbatim from the full version of this paper [CDKT19b].

This section presents three simple, intuitive, and—most importantly—practical PRNG constructions:

- a construction based on the *Merkle-Damgård paradigm* using a public *fixed-length compression function*;
- a construction based on the *Merkle-Damgård paradigm* using the *Davies-Meyer compression function* (as in SHA-2), which is built from any public block cipher; and
- a construction based on the *Sponge paradigm* (as in SHA-3), which uses a public permutation.

For PRNGs based on the MD paradigm, there are in fact two constructions: one achieving normal, computational PRNG security and one achieving information-theoretic (IT) security.

For the reader's convenience, Appendix 3.5 states the corresponding online extractor constructions along with the security bounds—for applications where extraction is sufficient.

5.1 PRNGs from Merkle-Damgård

5.1.1 Computational PRNGs from Merkle-Damgård

A PRNG can be obtained from a compression function F as follows (cf. Figure 5.1):[1]

Construction 10 (PRNG from Merkle-Damgård)**.** The (m, r)-PRNG construction $\mathsf{MD} = (\mathsf{refresh}, \mathsf{next})$ based on Merkle-Damgård with a compression function $F : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ is defined as follows:[2]

- $\mathsf{refresh}^F(s, x) = F(s, x)$, and
- $\mathsf{next}^F(s) = (F(s,0), F(s,1) \| \cdots \| F(s, r/n))$.

The security of Construction 10 is proved in the F-model, where F is a uniformly random function.

Theorem 5.1 (Robustness of Merkle-Damgård PRNGs)**.** *Construction 10 is a $(\gamma^*, q, t, \ell, \varepsilon_{\text{rob}})$-robust PRNG in the F-model for*

$$\varepsilon_{\text{rob}} \;\leq\; 2t \cdot \left(\frac{\tilde{q}^2 + \tilde{q}\ell + \ell^2}{2^n} + \frac{\tilde{q}}{2^{\gamma^*}} \right),$$

where $\tilde{q} = q + r/n + 1$.

Note that iteratively absorbing some input blocks $x_1, \ldots, x_\ell$ via refresh, starting with a state s_0 is identical to applying the Merkle-Damgård construction to the input with initialization vector (IV) s_0, which is denoted by $\mathsf{MD}^F_{s_0}(x_1, \ldots, x_\ell)$ in the remainder of this section.

As outlined in Section 4.2, the idea of the proof is to first establish extraction, maintaining, as well as next security, and to then show that extraction and next security imply recovering security and

[1] To reduce notational clutter, the algorithms refresh and next of the PRNG constructions are not "branded" with the design name. There will be no ambiguity as to which construction is meant in any place in this paper.

[2] The integer arguments to the compression function are to be naturally mapped to $\{0,1\}^n$.

that maintaining and next security imply preserving security. More precisely, the proof proceeds as follows:

- **Extraction security:** In order to establish extraction security using the H-coefficient method (cf. Theorem 3.1 in Section 3.1.1), one first defines *bad* transcripts as transcripts for which the attacker has queried F on all coordinates needed to evaluate the PRNG on the inputs it provided.

  For good transcripts, there is at least one missing F query to obtain the output of the PRNG. The probability ratio is lower bounded by arguing that the output of the PRNG looks random to the attacker unless there is a collision among the remaining F queries.

  The probability of bad transcripts is upper bounded in the *ideal* world ($b = 1$). The proof uses the legitimacy of the attacker in a resampling argument to argue that it is unlikely that the attacker makes all the queries necessary to evaluate the PRNG on the inputs it provides. Special care has to be taken to handle collisions in the F queries.

- **Maintaining security:** The corresponding proof also employs the H-coefficient method, but is considerably more straightforward. A *bad* transcript is a transcript for which the initial (random) state of the PRNG appears in the F-queries. The ratio is bounded analogously to the extraction proof, and bounding the probability of a bad transcript is trivial.

- **Next security:** This is again a simple H-coefficient proof, which amounts to showing that unless the attacker queries F on an input $(s, *)$, where s is the (random) state to which next is applied, the output of next looks random.

- **Recovering security:** Showing that the Merkle-Damgård construction achieves recovering security is a simple hybrid argument: First, one uses extraction security to argue that the state after absorbing the inputs can be replaced by a random value. Second, using next security, one argues that the output of next on said random value looks random.

- **Preserving security:** Proving preserving security is also a simple hybrid argument: First, one uses maintaining security to argue that the state after absorbing the inputs can be replaced by a random value. Second, using next security, one argues that the output of next on said random value looks random.

The final bound in Theorem 5.1 follows by applying Theorem 4.8.

5.1.1.1 Extraction Security

Lemma 5.2 (Extraction security)**.** *The advantage of any γ^*-legitimate (q, ℓ)-attacker $\mathcal{A}$ against extraction security of* MD *is bounded by*

$$\mathrm{Adv}^{\mathrm{ext},F}_{\mathsf{MD}}(\mathcal{A}) \;\leq\; \varepsilon^{\mathrm{ext}}_{\mathsf{MD}}(\gamma^*, q, \ell) \;:=\; \frac{q^2 + q\ell + \ell^2}{2^n} + \frac{2q}{2^{\gamma^*}} \,.$$

Proof. In order to prove Lemma 5.2 using the H-coefficient method, consider a transcript[3]

$$\tau \;=\; (s^*, s_0, x_1, \ldots, x_\ell, L)$$

of the interaction between the $\mathcal{A}$ and the extraction game, where s^* is the value returned by the game, L is the set of queries to the oracle made by the adversary $\mathcal{A}$, and where s_0 is the initial state and $x_1, \ldots, x_\ell$ the inputs provided by $\mathcal{A}$. Let $\ell' \geq 0$ be maximal such that

$$((y_{i-1}, x_i), y_i) \in L$$

for some values $y_0, y_1, \ldots, y_{\ell'}$ with $y_0 = s_0$. A transcript τ is called a *bad transcript* if $\ell' = \ell$; otherwise, τ is called *good*.

In order to apply Theorem 3.1, one merely needs to bound the probability ratio for good

[3] In order to keep notation simple, ℓ—here and in the following—is the number of inputs in a particular τ, not the upper bound from Lemma 5.2.

transcripts (Lemma 5.3) and the probability of a bad transcript occurring in the ideal world, i.e, for $b = 1$ (Lemma 5.4). □

Lemma 5.3 (Ratio analysis)**.** *For all* good *transcripts* τ*,*

$$\frac{\mathrm{p}_0(\tau)}{\mathrm{p}_1(\tau)} \;\geq\; 1 - \frac{q\ell + \ell^2}{2^n} \,.$$

Proof. Fix a good transcript τ and consider first $\mathrm{p}_1(\tau)$. Since in the ideal world s^* is sampled uniformly,

$$\mathrm{p}_1(\tau) \;=\; p_L \cdot 2^{-n} \,,$$

where p_L denotes the probability that a uniform random function is consistent with the queries in L. In the real world,

$$\mathrm{p}_0(\tau) \;=\; p_L \cdot q_\tau \,,$$

where q_τ is the probability that $\mathsf{MD}_{s_0}^{F_L}(x_1, x_2, \ldots, x_\ell) \;=\; s^*$ over a function F_L that is sampled uniformly at random conditioned on being consistent with L.

It remains to derive a lower bound on q_τ. To that end, observe that due to τ being a good transcript, $\ell' < \ell$. Hence, q_τ is the probability (over F_L) that

$$Y_\ell \;:=\; \mathsf{MD}_{y_{\ell'}}^{F_L}(x_{\ell'+1}, \ldots, x_\ell) \;=\; s^* \,.$$

Consider the intermediate chaining values $Y_{\ell'+1}, \ldots, Y_{\ell-1}$ and, for $i = \ell' + 1, \ldots, \ell - 1$, define the event $\mathsf{FRESH_i}$ that Y_i is *fresh*, i.e., there is no query $((Y_i, *), *) \in L$ and $Y_i \neq Y_j$ for $j < i$. Let,

$$\mathsf{FRESH} \;:=\; \bigcap_{i=\ell'+1}^{\ell-1} \mathsf{FRESH_i} \,.$$

Then,

$$\Pr[Y_\ell = s^* | \mathsf{FRESH}] \;=\; 2^{-n}$$

since the conditioning implies that $F(Y_{\ell-1}, x_\ell)$ is freshly sampled and, hence, Y_ℓ is a uniformly random value. Moreover,

$$\Pr\left[\overline{\mathsf{FRESH_i}} \,\middle|\, \bigcap_{k=\ell'+1}^{i-1} \mathsf{FRESH_k}\right] \;\leq\; \frac{q+\ell}{2^n}$$

as there are at most $q+\ell$ non-fresh values by the time Y_i is sampled. Here, the notation $\overline{A}$ denotes the complement of an event A. Therefore,

$$\Pr[\mathsf{FRESH}] \;\geq\; 1 - \frac{q\ell + \ell^2}{2^n}\,,$$

and, finally,

$$\begin{aligned} q_\tau \;&\geq\; \Pr[\mathsf{FRESH}] \cdot \Pr[Y_\ell = s^* | \mathsf{FRESH}] \\ &\geq\; \left(1 - \frac{q\ell + \ell^2}{2^n}\right) \cdot 2^{-n}\,, \end{aligned}$$

which implies

$$\frac{\mathsf{p}_0(\tau)}{\mathsf{p}_1(\tau)} \;\geq\; 1 - \frac{q\ell + \ell^2}{2^n}\,.$$

□

Lemma 5.4 (Bad event analysis)**.** *For the set $\mathcal{B}$ of bad transcripts (as defined above),*

$$\Pr[T_1 \in \mathcal{B}] \;\leq\; \frac{2q}{2^{\gamma^*}} + \frac{q^2}{2^n}\,.$$

Proof. Observe that in the ideal world, the output of the extraction game is a uniformly random value s^*, which is independent of the initial state S_0 and the inputs $X_1, \ldots, X_\ell$. The sampling order of the ideal experiment can therefore be changed to be the following:

1. Sample F uniformly at random.

2. Run $\mathcal{A}$ until it outputs $(\tilde{s}_0, \tilde{x}_1, \ldots, \tilde{x}_{\tilde{\ell}})$, thereby also generating the state σ and the list of queries L_0 before the challenge.

3. Choose s^* uniformly at random.

4. Continue running $\mathcal{A}$ on σ and s^*, letting it make additional queries L_1.

5. Resample the inputs $(X_1, \ldots, X_\ell)$ conditioned on $(\Sigma, \mathcal{L}_0, S_0) = (\sigma, L_0, s_0)$.

Remember that $\mathcal{L}$ is the random variable for the set of queries made while L is a particular value that the random variable takes. Observe that since the conditioning includes $\mathcal{L}_0$, $\mathcal{A}$ makes the same queries, L_0, during the first run and the resampling process. Moreover, since conditioned on $(\Sigma, \mathcal{L}_0, S_0)$, $(X_1, \ldots, X_\ell)$ and $\mathcal{L}_1$ are independent, the min-entropy condition holds for $\mathcal{L} = \mathcal{L}_0 \cup \mathcal{L}_1$, the list of all queries made by $\mathcal{A}$ during the experiment, as well. That is,

$$\mathrm{H}_\infty(X_1, \ldots, X_\ell | \Sigma \mathcal{L} S_0) \;\geq\; \gamma^* \,.$$

Let $\mathcal{E}$ be the event that there exists a collision in $\mathcal{L}$, i.e., two or more queries have the same answer. Note that

$$\Pr[T_1 \in \mathcal{B}] \;\leq\; \Pr[T_1 \in \mathcal{B} | \mathcal{E}] + \Pr[\overline{\mathcal{E}}] \;\leq\; \Pr[T_1 \in \mathcal{B} | \mathcal{E}] + \frac{q^2}{2^n} \,.$$

Towards bounding $\Pr[T_1 \in \mathcal{B} | \mathcal{E}]$, consider now a particular triple $z = (\sigma, L, s_0) \in \mathcal{E}$, which is shorthand for L being collision-free. Define a *potential chain* as values $y_0, y_1, \ldots, y_\ell$ for *some* ℓ such that $y_0 = s_0$ and

$$((y_{i-1}, v_i), y_i) \in L \,.$$

for $i = 1, \ldots, \ell$ and some values $v_1, \ldots, v_\ell$. Observe that without collisions, L can contain at most q potential chains. Clearly, conditioned on $Z = z$, $T_1 \in \mathcal{B}$ if and only if for some potential chain,

$X_i = v_i$ for all $i = 1, \ldots, \ell$. Hence, by the legitimacy of $\mathcal{A}$,

$$\Pr[T_1 \in \mathcal{B} | Z = z] \;\leq\; q \cdot p_z \,,$$

where $p_z := \mathsf{Pred}(\overline{X}|Z = z)$. In expectation,

$$\begin{aligned}
\Pr[T_1 \in \mathcal{B}|\mathcal{E}] \;&=\; \sum_{z \in \mathcal{E}} \Pr[Z = z|\mathcal{E}] \cdot \Pr[T_1 \in \mathcal{B}|Z = z] \\
&\leq\; \sum_{z \in \mathcal{E}} \Pr[Z = z|\mathcal{E}] \cdot q \cdot p_z \\
&=\; q \cdot \mathsf{Pred}(\overline{X}|Z\mathcal{E}) \\
&\leq\; q \cdot \frac{\mathsf{Pred}(\overline{X}|Z)}{1 - q^2/2^n} \\
&\leq\; \frac{q}{(1 - q^2/2^n) \cdot 2^{\gamma^*}} \;\leq\; \frac{2q}{2^{\gamma^*}} \,,
\end{aligned}$$

using that[4] $q^2/2^n \leq 1/2$ and where the penultimate inequality is due to

$$\begin{aligned}
\mathsf{Pred}(X|Z) \;&\geq\; \sum_{z \in \mathcal{E}} \Pr[Z = z] \cdot p_z \\
&=\; \Pr[\mathcal{E}] \cdot \sum_{z \in \mathcal{E}} \frac{\Pr[Z = z]}{\Pr[\mathcal{E}]} \cdot p_z \\
&=\; \Pr[\mathcal{E}] \cdot \mathsf{Pred}(\overline{X}|Z\mathcal{E}) \,.
\end{aligned}$$

□

[4] This can always be assumed as the bound would otherwise be vacuous.

5.1.1.2 Maintaining Security

Lemma 5.5 (Maintaining security). *The advantage of any (q, ℓ)-attacker $\mathcal{A}$ against maintaining security is bounded by*

$$\mathrm{Adv}^{\mathrm{mtn},F}_{\mathsf{MD}}(\mathcal{A}) \;\le\; \varepsilon^{\mathrm{mtn}}_{\mathsf{MD}}(q,\ell) \;:=\; \frac{q\ell + \ell^2}{2^n} + \frac{q}{2^n}\,.$$

Proof. To bound the advantage of an attacker $\mathcal{A}$ at guessing b via an H-coefficient proof, consider a transcript

$$\tau = (s^*, s_0, x_1, \ldots, x_\ell, L)$$

between $\mathcal{A}$ and the maintaining game, where s^* is the output of the game and L are the queries made by $\mathcal{A}$. A transcript is *bad* if there is a query of the type $((s_0, *), *) \in L$ and *good* otherwise. The probability of a bad transcript occurring in the $b = 1$ case is clearly at most $|L|/2^n$. Moreover, for good transcripts τ,

$$\mathrm{p}_1(\tau) \;=\; 2^{-2n} \cdot p_L\,,$$

where p_L denotes the probability that a uniform random function is consistent with the queries in L. Furthermore, by an argument similar to that in the proof of Lemma 5.3,

$$\mathrm{p}_0(\tau) \;\ge\; \left(1 - \frac{q\ell + \ell^2}{2^n}\right) \cdot 2^{-2n} \cdot p_L\,.$$

The lemma follows by applying Theorem 3.1. □

5.1.1.3 Next Security

Lemma 5.6 (Next security). *The advantage of any q-attacker $\mathcal{A}$ against next security is bounded by*

$$\mathrm{Adv}^{\mathrm{next},F}_{\mathsf{MD}}(\mathcal{A}) \;\le\; \varepsilon^{\mathrm{next}}_{\mathsf{MD}}(q) \;:=\; \frac{q}{2^n}\,.$$

Proof. For a straight-forward H-coefficient proof, consider the transcript

$$\tau = (s, s^*, L) \ ,$$

where s is the initial state, s^* is the input given to $\mathcal{A}$, and L are the queries $\mathcal{A}$ makes to F. A transcript is *bad* if L contains a query of the form $((s, *), *)$ and *good* otherwise. It is easily seen that for good transcripts, the probability ratio is 1, whereas, in the ideal world, where $\mathcal{A}$'s view is completely independent of S, the probability of a bad event is at most $q/2^n$. □

5.1.1.4 Recovering Security

In the following, let $\varepsilon^{\text{ext}}_{\mathsf{MD}}(\gamma^*, q, \ell)$ and $\varepsilon^{\text{next}}_{\mathsf{MD}}(q)$ be as in Lemmas 5.2 and 5.6. Extraction and next security together imply recovering security:

Lemma 5.7 (Recovering Security)**.** *For every γ^*-legitimate (q, ℓ)-attacker $\mathcal{A}$,*

$$\text{Adv}^{\text{rec},F}_{\mathsf{MD}}(\mathcal{A}) \ \leq \ \varepsilon^{\text{ext}}_{\mathsf{MD}}(\gamma^*, q + r/n + 1, \ell) + \varepsilon^{\text{next}}_{\mathsf{MD}}(q + r/n + 1) \ .$$

Proof. For $b \in \{0, 1\}$, denote by H_b the recovering experiment conditioned on the secret bit having the value b. Moreover, define a hybrid experiment $H_{\frac{1}{2}}$ in which the challenge oracle returns $\text{next}^F(U_n)$ to $\mathcal{A}$. By the triangle inequality, to prove the lemma, it suffices to bound the distance between experiments H_0 and $H_{\frac{1}{2}}$ and $H_{\frac{1}{2}}$ and H_1.

- Towards bounding the distance between H_0 and $H_{\frac{1}{2}}$, consider the following attacker $\mathcal{A}_{\text{ext}}$ against extraction security of MD: $\mathcal{A}_{\text{ext}}$ runs $\mathcal{A}$ answering its oracle queries by passing queries to F. At some point, $\mathcal{A}$ outputs $(s_0, x_1, ..., x_\ell)$. $\mathcal{A}_{\text{ext}}$ forwards $(s_0, x_1, x_2, ..., x_\ell)$ to the challenger. $\mathcal{A}_{\text{ext}}$ receives s^* as response from the challenger. $\mathcal{A}_{\text{ext}}$ now computes next on the input s^*, by making $r/n + 1$ additional queries to the primitive, on the inputs (s^*, i) for $i = 0, 1, \ldots, r/n$. It forwards to $\mathcal{A}$ the values $(F(s^*, 0), F(s^*, 1)|| \ldots ||F(s^*, r/n))$. It proceeds

to respond to $\mathcal{A}$'s oracle calls as before and waits for $\mathcal{A}$'s guess bit. It merely forwards the same bit as its guess to the challenger.

When the challenger's bit is 0, $\mathcal{A}_{\text{ext}}$ perfectly simulates towards $\mathcal{A}$ the distribution H_0. When the challenger's bit is 1, $\mathcal{A}$ is given $\mathsf{next}(U_n)$. This corresponds to the hybrid distribution $H_{\frac{1}{2}}$. Moreover, it is easily seen that if $\mathcal{A}$ is γ^*-legitimate, so is $\mathcal{A}_{\text{ext}}$. Thus, the advantage of $\mathcal{A}$ in distinguishing hybrids H_0 and $H_{\frac{1}{2}}$ is upper-bounded by the advantage of $\mathcal{A}_{\text{ext}}$ in the extraction game which is $\varepsilon^{\text{ext}}_{\mathsf{MD}}(\gamma^*, q + r/n + 1, \ell)$.

- Towards bounding the distance between $H_{\frac{1}{2}}$ and H_1, consider the following attacker $\mathcal{A}_{\text{next}}$ against next security of MD: $\mathcal{A}_{\text{next}}$ runs $\mathcal{A}$ answering its oracle queries by passing queries to F. When $\mathcal{A}$ outputs $(s_0, x_1, ..., x_\ell)$, $\mathcal{A}_{\text{next}}$ returns its own distinguishing challenge, continuing to answer oracle queries for $\mathcal{A}$. In the end, it outputs $\mathcal{A}$'s guess bit.

  It is straight-forward to verify that $\mathcal{A}_{\text{next}}$ perfectly simulates $H_{\frac{1}{2}}$ to $\mathcal{A}$ when given $\mathsf{next}^F(U_n)$ and H_1 when given U_{n+r}. Thus, the advantage of $\mathcal{A}$ in distinguishing hybrids $H_{\frac{1}{2}}$ and H_1 is upper-bounded by the advantage of $\mathcal{A}_{\text{next}}$ in the next game, which is $\varepsilon^{\text{next}}_{\mathsf{MD}}(q + r/n + 1)$.

□

5.1.1.5 Preserving Security

In the following, let $\varepsilon^{\text{mtn}}_{\mathsf{MD}}(q, \ell)$ and $\varepsilon^{\text{next}}_{\mathsf{MD}}(q)$ be as in Lemmas 5.5 and 5.6. Maintaining and next security together imply preserving security:

Lemma 5.8 (Preserving Security)**.** *For every (q, ℓ)-attacker $\mathcal{A}$,*

$$\mathrm{Adv}^{\text{pre},F}_{\mathsf{MD}}(\mathcal{A}) \;\leq\; \varepsilon^{\text{mtn}}_{\mathsf{MD}}(q + r/n + 1, \ell) + \varepsilon^{\text{next}}_{\mathsf{MD}}(q + r/n + 1)\ .$$

Proof. For $b \in \{0, 1\}$, denote by H_b the preserving experiment conditioned on the secret bit having the value b. Moreover, define a hybrid experiment $H_{\frac{1}{2}}$ in which the challenge oracle returns

$\mathsf{next}^F(U_n)$ to $\mathcal{A}$. By the triangle inequality, to prove the lemma, it suffices to bound the distance between experiments H_0 and $H_{\frac{1}{2}}$ and $H_{\frac{1}{2}}$ and H_1.

- Towards bounding the distance between H_0 and $H_{\frac{1}{2}}$, consider the following attacker $\mathcal{A}_{\text{mtn}}$ against maintaining security of MD: $\mathcal{A}_{\text{mtn}}$ runs $\mathcal{A}$ answering its oracle queries by passing queries to F. At some point, $\mathcal{A}$ outputs $(x_1, ..., x_\ell)$. $\mathcal{A}_{\text{mtn}}$ forwards $(x_1, x_2, ..., x_\ell)$ to the challenger. $\mathcal{A}_{\text{mtn}}$ receives s^* as response from the challenger. $\mathcal{A}_{\text{mtn}}$ now computes next on the input s^*, by making $r/n + 1$ additional queries to the primitive, on the inputs (s^*, i) for $i = 0, 1, \ldots, r/n$. It forwards to $\mathcal{A}$ the values $(F(s^*, 0), F(s^*, 1)|| \ldots ||F(s^*, r/n))$. It proceeds to respond to $\mathcal{A}$'s oracle calls as before and waits for $\mathcal{A}$'s guess bit. It merely forwards the same bit as its guess to the challenger.

  When the challenger's bit is 0, the $\mathcal{A}_{\text{mtn}}$ perfectly simulates towards $\mathcal{A}$ the distribution H_0. When the challenger's bit is 1, $\mathcal{A}$ is given $\mathsf{next}(U_n)$. This corresponds to the hybrid distribution $H_{\frac{1}{2}}$. Thus, the advantage of $\mathcal{A}$ in distinguishing hybrids H_0 and $H_{\frac{1}{2}}$ is upper-bounded by the advantage of $\mathcal{A}_{\text{mtn}}$ in the extraction game which is $\varepsilon^{\text{mtn}}_{\mathsf{MD}}(q + r/n + 1, \ell)$.

- Towards bounding the distance between $H_{\frac{1}{2}}$ and H_1, consider the following attacker $\mathcal{A}_{\text{next}}$ against next security of MD: $\mathcal{A}_{\text{next}}$ runs $\mathcal{A}$ answering its oracle queries by passing queries to F. When $\mathcal{A}$ outputs $(x_1, ..., x_\ell)$, $\mathcal{A}_{\text{next}}$ returns its own distinguishing challenge, continuing to answer oracle queries for $\mathcal{A}$. In the end, it outputs $\mathcal{A}$'s guess bit.

  It is straight-forward to verify that $\mathcal{A}_{\text{next}}$ perfectly simulates $H_{\frac{1}{2}}$ to $\mathcal{A}$ when given $\mathsf{next}^F(U_n)$ and H_1 when given U_{n+r}. Thus, the advantage of $\mathcal{A}$ in distinguishing hybrids $H_0, H_{\frac{1}{2}}$ is upper-bounded by the advantage of $\mathcal{A}_{\text{next}}$ in the next game which is $\varepsilon^{\text{next}}_{\mathsf{MD}}(q + r/n + 1)$.

□

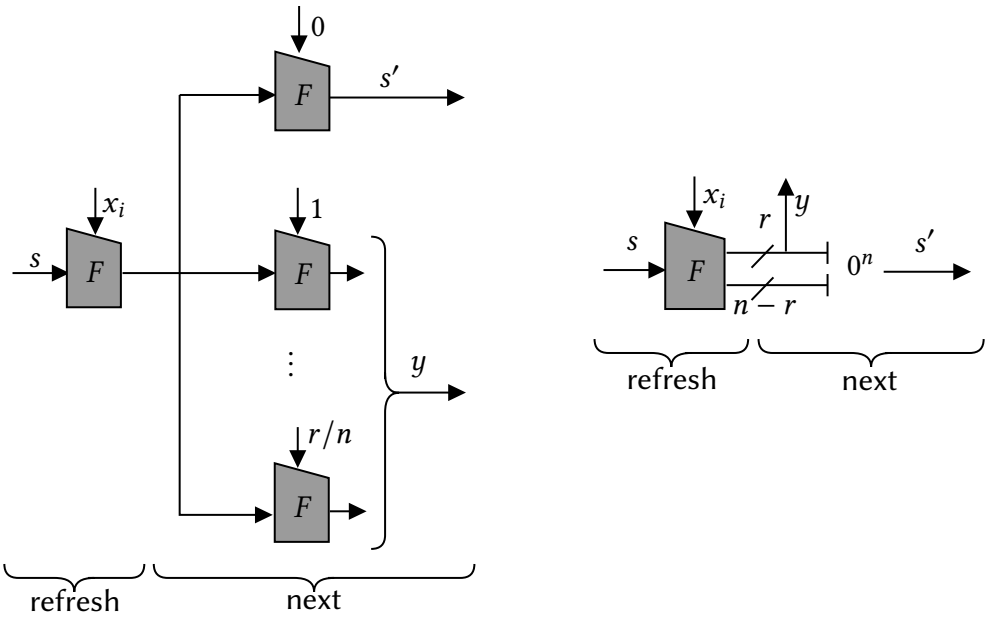

Figure 5.1: Procedures refresh (processing a single-block input x_i) and next of Merkle-Damgård PRNG constructions with compression function F. Left: Computationally secure Construction 10; right: IT secure Construction 11.

5.1.2 IT PRNGs from Merkle-Damgård

An IT-robust PRNG based on Merkle-Damgård can be obtained if the next function simply outputs the truncated state (and outputs 0^n as the new state):

Construction 11 (IT-PRNG from Merkle-Damgård)**.** The (m, r)-PRNG construction $\mathsf{MD}_r = (\mathsf{refresh}, \mathsf{next})$ based on Merkle-Damgård with a compression function $F : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ is defined as follows:

- $\mathsf{refresh}^F(s, x) = F(s, x)$, and
- $\mathsf{next}^F(s) = (0^n, s[1..r])$.

The security of Construction 11 is proved in the F-model, where F is a uniformly random function.

To state the theorem for the IT construction, for an integer ℓ, let

$$d'(\ell) = \max_{\ell' \in \{1,\ldots,\ell\}} |\{d \in \mathbb{N} \ : \ d|\ell'\}| \, .$$

Observe that, asymptotically, $d'(\ell)$ grows very slowly, i.e., as $\ell^{o(1)}$. Furthermore, let F be a random compression function.

Theorem 5.9 (IT-Robustness of Merkle-Damgård PRNGs)**.** *Construction 11 is a* $(\gamma^*, q, t, \ell, \varepsilon_{\text{rob}})$*-IT-robust PRNG in the* F*-model, where*

$$\varepsilon_{\text{rob-it}} \ \leq \ \frac{t}{2}\sqrt{\frac{2^{r-\gamma^*}}{(1-\rho)} + \ell \cdot d'(\ell) \cdot \frac{2^r}{2^n} + 64\ell^4 \cdot \frac{2^r}{2^{2n}} + 16\ell^2 \cdot \frac{\tilde{q}^2 2^r}{2^{2n}}} \ + \ t\rho \, ,$$

for $\rho = \frac{\tilde{q}^2}{2^r}$ *where* $\tilde{q} = q + t\ell$*.*

The theorem will be a direct consequence of the following lemma, and Theorem 4.12 above, which adds a multiplicative factor t to obtain the bound in the theorem.

Lemma 5.10. *For every* γ^**-IT-legitimate* (q, ℓ)*-attacker in the ideal compression function model,*

$$\text{Adv}^{\text{rec-IT},\gamma^*}_{\text{MD}_r}(\mathcal{A}) \ \leq \ \frac{1}{2}\sqrt{\frac{2^{r-\gamma^*}}{(1-\rho)} + \ell \cdot d'(\ell) \cdot \frac{2^r}{2^n} + 64\ell^4 \cdot \frac{2^r}{2^{2n}} + 16\ell^2 \frac{q^2 2^r}{2^{2n}}} + \rho \ \ ,$$

where $\rho = \frac{q^2}{2^r}$*.*

Before we turn to a proof, we note the challenges here. In particular, the core of the proof will be to show a bound on the collision probability of the output of next for a random compression function F, which in turn will reduce to outputting the truncation of the output of the MD construction. This problem resembles that studied by [DGH+04] and by [GPR14]. However, the difficulty here is that we need to consider the set of queries previously done by the adversary $\mathcal{A}$. This will significantly complicate the proof—we also modify slightly the graph-theoretic for-

malization adopted by these previous works to something more amenable to our more complex setting.

Proof of Lemma 5.10. We define a few random variables which we will be using in our proofs.

- F a randomly chosen compression function, to which the adversary is given access. We use F both for the oracle itself, as well as for the random variable describing the entire function table.
- ℓ: Number of blocks input to the challenge oracle (which is a random variable itself, we overload notation here, using the same letter we use in the bound on ℓ in the lemma statement).
- $\overline{X} = (\overline{X}_1, \ldots, \overline{X}_\ell)$: the blocks input to the challenge oracle.
- $\tilde{Y}_\ell$: output of MD_r. Let us remind ourselves that this is s truncated to r bits (the output of next);
- $Z = (\Sigma, \mathcal{L}, S_0)$: "side information" where Σ is the attacker state before challenge, $\mathcal{L}$ is the attacker query/answers to F before challenge , S_0 is the initial PRNG state provided by $\mathcal{A}$;
- U_r: uniform r-bit string.

The advantage of the adversary $\mathcal{A}$ in the recovering game is bounded by $\mathsf{SD}\left((\tilde{Y}_\ell, Z, F), (U_r, Z, F)\right)$. Therefore, it is sufficient to upper-bound that. The reasoning is quite simple. Note that Z contains the state of the attacker Σ just before it makes the challenge query. This means that $\mathcal{A}$ cannot tell apart real from random.

We also define an event $\mathcal{E}$ where the answers to the F-queries by $\mathcal{A}$ are distinct *when truncated to the first r bits*. Note that there are a maximum of q queries in this list. We would like to point out that $\mathcal{E}$ has the same probability of occurring in either experiment, since the experiments are

identical up to the point when this even is defined. Therefore, by Proposition 3.4,

$$\mathsf{SD}\left((\tilde{Y}_\ell, Z, F), (U_r, Z, F)\right) \;\leq\; \mathsf{SD}\left((\tilde{Y}_\ell, Z, F)|\mathcal{E}, (U_r, Z, F)|\mathcal{E}\right) + \frac{q^2}{2^r}\,;$$

For convenience we let $\rho := q^2/2^r$ for the remainder of the proof. In order to bound the statistical distance conditioned on $\mathcal{E}$, we can rewrite the same as as

$$\mathsf{SD}\left((\tilde{Y}_\ell, Z, F)|\mathcal{E}, (U_r, Z, F)|\mathcal{E}\right) \;=\; \sum_{z\in\mathcal{E}} \Pr[Z = z|\mathcal{E}] \cdot \mathsf{SD}\left((\tilde{Y}_\ell, F)|z, (U_r, F)|z\right)\,, \tag{5.1}$$

where $z \in \mathcal{E}$ is to denote that the sum is taken over all side informations $Z = z$, satisfying $\mathcal{E}$.[5] Define $p_z := \mathsf{Pred}(\overline{X}|Z = z)$, and observe that

$$\mathbf{E}_z[p_z] \;=\; \mathsf{Pred}(\overline{X}|Z) \;\leq\; 2^{-\gamma^*}\,,$$

where the latter inequality follows from the assumption $\mathrm{H}_\infty(\overline{X}|Z) \geq \gamma^*$. Moreover,

$$\mathrm{H}_\infty(\overline{X}|Z\mathcal{E}) \;\geq\; \gamma^* - \log(1-\rho)^{-1}\,,$$

which is due to

$$\begin{aligned}
\mathsf{Pred}(\overline{X}|Z) \;&\geq\; \sum_{z\in\mathcal{E}} \Pr[Z = z] \cdot p_z \\
&=\; \Pr[\mathcal{E}] \cdot \sum_{z\in\mathcal{E}} \frac{\Pr[Z = z]}{\Pr[\mathcal{E}]} \cdot p_z \\
&=\; \Pr[\mathcal{E}] \cdot \mathsf{Pred}(\overline{X}|Z\mathcal{E})\,.
\end{aligned}$$

[5] Therefore $\mathcal{E}$ can be omitted in the conditioning of the statistical distance.

From Lemma 5.11 we prove below, we will get,

$$\mathsf{SD}\left((\tilde{Y}_\ell, F)|z, (U_r, F)|z\right) \;\leq\; \frac{1}{2}\sqrt{2^r p_z + \ell \cdot d'(\ell) \cdot \frac{2^r}{2^n} + 64\ell^4 \cdot \frac{2^r}{2^{2n}} + 16\ell^2 q^2 \cdot \frac{2^r}{2^{2n}}}\;.$$

Using Jensen's inequality, (5.1) becomes, for $\alpha = 2^r$ and $\beta = \ell \cdot d'(\ell) \cdot \frac{2^r}{2^n} + 64\ell^4 \cdot \frac{2^r}{2^{2n}} + 16\ell^2 q^2 \cdot \frac{2^r}{2^{2n}}$,

$$\begin{aligned}\mathsf{SD}\left((\tilde{Y}_\ell, F)|\mathcal{E}, (U_r, F)|\mathcal{E}\right) &\leq \frac{1}{2}\sqrt{\alpha \mathsf{Pred}(\overline{X}|\mathcal{L}\mathcal{E}) + \beta} \\ &\leq \frac{1}{2}\sqrt{\frac{2^{r-\gamma^*}}{(1-\rho)} + \ell \cdot d'(\ell) \cdot \frac{2^r}{2^n} + 64\ell^4 \cdot \frac{2^r}{2^{2n}} + 16\ell^2 \frac{q^2 2^r}{2^{2n}}}\;.\end{aligned}$$

□

Lemma 5.11. *For $z \in \mathcal{E}$ and the random variables as defined earlier,*

$$\mathsf{SD}\left((\tilde{Y}_\ell, F)|z, (U_r, F)|z\right) \;\leq\; \frac{1}{2}\sqrt{2^r p_z + \ell \cdot d'(\ell) \cdot \frac{2^r}{2^n} + 64\ell^4 \cdot \frac{2^r}{2^{2n}} + 16\ell^2 q^2 \cdot \frac{2^r}{2^{2n}}}\;.$$

Proof. Fix $z = (\sigma, L, s_0)$. Observe that, conditioned on z, F is distributed uniformly over the set of all functions that agree with L. Thus, by Proposition 3.3,

$$\mathsf{SD}\left((\tilde{Y}_\ell, F)|z, (U_r, F)|z\right) \;\leq\; \frac{1}{2}\sqrt{2^r \cdot \mathsf{Coll}(\tilde{Y}_\ell|Fz) - 1}\;. \tag{5.2}$$

To bound the collision probability, we consider the following experiment:

- choose F uniformly consistent with L
- sample inputs $\overline{X} = (\overline{X}_1, \ldots, \overline{X}_\ell)$ and $\overline{X}' = (\overline{X}'_1, \ldots, \overline{X}'_{\ell'})$ independently but conditioned on $Z = z$.
- compute $\tilde{Y}_\ell$ and $\tilde{Y}'_\ell$ as the truncated MD evaluations with F of $\overline{X}$ and $\overline{X}'$

Then, we bound the probability that $\tilde{Y}_\ell = \tilde{Y}'_\ell$ in this experiment as

$$\Pr[\tilde{Y}_\ell = \tilde{Y}'_\ell] \;\leq\; \Pr[\overline{X} = \overline{X}'] + \Pr[\tilde{Y}_\ell = \tilde{Y}'_\ell | \overline{X} \neq \overline{X}'] \;. \tag{5.3}$$

Clearly, the former is at most p_z. To bound the latter, we fix arbitrary inputs $\overline{x} \neq \overline{x}'$ of lengths ℓ and ℓ', respectively. We also assume, wlog, that the evaluation of $\overline{x}'$ is not completely covered by L; due to the collision-freeness of L. Let $\overline{x}_{k+1}$ be the first block of $\overline{x}$ not covered by $\mathcal{L}$ and similarly $\overline{x}'_{k+1}$ for $\overline{x}'$. We let $k = \ell$ if all blocks are covered.

The structure graph. We will now model the evaluation producing $\tilde{Y}_\ell$ and $\tilde{Y}'_\ell$ as a (labeled) graph process. In particular, let us define for convenience

$$x^{(i)} := \begin{cases} \overline{x}_i & i \leq \ell \\ \overline{x}'_{(i-\ell)} & \text{otherwise} \end{cases}$$

for all $i = 1, \ldots, \ell + \ell'$, and we let $\tilde{\ell} = \ell + \ell'$.

For a given compression function F (consistent with L), first define a labeled (multi-)graph $H_F(\overline{x}, \overline{x}') = (\mathcal{V}, \mathcal{E}.\mathcal{L})$ vertex set $\mathcal{V} \subseteq \{0,1\}^n$. Each edge will have a label $\mathcal{L}(e) = (x, c)$ – where $x \in \{0,1\}^m$ and $c \in \{\text{red}, \text{blue}\}$. Now we add edges as follows, allowing replications of edges (we will then explain below how to remove duplicates). We will implicitly define $\mathcal{V}$ as the set of n-bit strings which are endpoint to an edge:

Blue edges. For every $((x, y), y')$ in L we add an edge (y, y') with label (x, blue), and refer to such edges as the *blue* edges.

Red edges. Define

$$s_i := \begin{cases} 0^n & i = 0 \\ F(s_{i-1}, \overline{x}_i) & 1 \le i \le \ell \\ F(0^n, \overline{x}'_1) & i = \ell + 1 \\ F\left(s_{i-1}, \overline{x}'_{(i-\ell)}\right) & \ell + 2 \le i \le \tilde{\ell}\ . \end{cases}$$

Now for $i = 1, \ldots, \ell$, we add edge (s_{i-1}, s_i) with label $(\overline{x}_i, \text{red})$, unless the edge (s_{i-1}, s_i) is already present with label $(\overline{x}_i, c)$ for $c \in \{\text{blue}, \text{red}\}$. We add the edge $(0^n, s_{\ell+1})$ with label $(\overline{x}'_1, \text{red})$, unless the edge (s_{i-1}, s_i) is already present with label $(\overline{x}'_1, c)$ for $c \in \{\text{blue}, \text{red}\}$. Finally, we add each (s_{i-1}, s_i) for $i = \ell+2, \ldots, \tilde{\ell}$ with label $(\overline{x}'_i, \text{red})$, unless the edge (s_{i-1}, s_i) is already present with label $(\overline{x}'_i, c)$ for $c \in \{\text{blue}, \text{red}\}$. We refer to these edges we added as the *red* edges.

Note in particular that we may have two identical edges e_1 and e_2, but in this case they will have labels (x_1, c_1) and (x_2, c_2) with $x_1 \neq x_2$, and moreover, at least one of them is red by our assumption on L.

Definition 5.12. The *structure graph* of $G_F = G_F(\overline{x}, \overline{x}') = (\mathcal{V}, \mathcal{E}, \mathcal{L})$ is the graph obtained from H_F as follows. We first look at all isomorphic graphs to G_F with vertices $\{0, \ldots, |\mathcal{V}| - 1\}$ such that 0^n is mapped to 0. We then pick the lexicographically first such graph.[6]

Let $\mathcal{G}_L(\overline{x}, \overline{x}')$ be the set of all $G_F(\overline{x}, \overline{x}')$ for an F compatible with L. Note that G_F will have two (possibly overlapping) paths starting from 0 with edges labeled by $\overline{x}$ and $\overline{x}'$, containing blue and red edges. We say that G_F is *colliding* if these two paths end in the same vertex, and let $\text{Coll}_L(\overline{x}, \overline{x}') \subseteq \mathcal{G}_L(\overline{x}, \overline{x}')$ be the set of colliding structure graphs.

Definition 5.13 (Accidents)**.** Let now $\mathcal{B}$ be the set of vertices of the sub-graph of G_F induced by the blue edges, plus 0. We now traverse the path induced by $\overline{x}$, and then the path induced by $\overline{x}'$.

[6] Indeed, the actual labeling of graphs will not matter below.

Each time we encounter a vertex which is not in $\mathcal{B}$, we add it to it. Each time we encounter a vertex which is *already* in $\mathcal{B}$, we say that an *accident* has occurred. We let $\mathsf{Acc}(G_F)$ be the number of accidents in G_F.

We also let $\mathcal{G}_L^a(\overline{x}, \overline{x}')$ to be the set of $H \in \mathcal{G}_L(\overline{x}, \overline{x}')$ with $\mathsf{Acc}(H) = a$. We state the following lemmas.

Lemma 5.14. *Let F be sampled randomly consistent with L, and $\overline{x} \neq \overline{x}'$. Then, let $\tilde{Y}_\ell$ and $\tilde{Y}'_\ell$ be values obtained after truncating at the end of the MD evaluations on inputs $\overline{x}$ and $\overline{x}'$ respectively. Then,*

$$\Pr[\tilde{Y}_\ell = \tilde{Y}'_\ell] \leq \Pr[G_F(\overline{x}, \overline{x}') \in \mathsf{Coll}_L(\overline{x}, \overline{x}')] + \frac{1}{2^r} .$$

Proof. We rewrite $\Pr[\tilde{Y}_\ell = \tilde{Y}'_\ell]$ as:

$$\Pr[\tilde{Y}_\ell = \tilde{Y}'_\ell] \leq \Pr[G_F(\overline{x}, \overline{x}') \in \mathsf{Coll}_L(\overline{x}, \overline{x}')] + \Pr[\tilde{Y}_\ell = \tilde{Y}'_\ell | G_F(\overline{x}, \overline{x}') \notin \mathsf{Coll}_L(\overline{x}, \overline{x}')]$$

We take a closer look at the latter term. Note that the graph is a fixed graph and it has no collision at the end nodes. We proceed to assign random, yet distinct values to the vertices. These are chosen from $\{0,1\}^n$. Note that it is sufficient to look at the two output vertices *locally* without looking at the global state. There are $2^n(2^n - 1)$ pairs of values for these output vertices such that these values are distinct. However, of these, $2^r 2^{n-r}(2^{n-r} - 1)$ have values which are equal in the first r bits and yet are distinct n-bit strings. Therefore,

$$\begin{aligned}
\Pr[\tilde{Y}_\ell = \tilde{Y}'_\ell | G_F(\overline{x}, \overline{x}') \notin \mathsf{Coll}_L(\overline{x}, \overline{x}')] &\leq \frac{2^r 2^{n-r}(2^{n-r} - 1)}{2^n(2^n - 1)} \\
&= \frac{2^{n-r} - 1}{2^n - 1} \\
&\leq \frac{2^{n-r}}{2^n} = \frac{1}{2^r} .
\end{aligned}$$

Putting it together, we have:

$$\Pr[\tilde{Y}_\ell = \tilde{Y}'_\ell] \leq \Pr[G_F(\overline{x}, \overline{x}') \in \mathsf{Coll}_L(\overline{x}, \overline{x}')] + \frac{1}{2^r} \ . \tag{5.4}$$

□

Lemma 5.15. *Let F be sampled randomly consistent with L, and $\overline{x} \neq \overline{x}'$. Let $H \in \mathcal{G}_L(\overline{x}, \overline{x}')$. Then,*

$$\Pr[G_F(\overline{x}, \overline{x}') = H] = \frac{1}{2^{n \cdot \mathsf{Acc}(H)}} \ .$$

Proof. We have two messages $\overline{x} \neq \overline{x}'$ and let $x^{(i)}$ be as defined before. We have a randomly sampled F which is consistent with L. Let us assume that the values $S_1, \ldots, S_{\tilde{\ell}}$ are revealed to us stepwise. S_i is the random variable representing the vertex after block i. Let $G_F = G_F(\overline{x}, \overline{x}')$. We define a consistency notion as follows: G_F is consistent with a given graph H after step $i \leq \tilde{\ell}$, denoted by Cons_i if the structure graphs $G_F^{(i)}$ and $H^{(i)}$ are equal as triples $(\mathcal{V}, \mathcal{E}.\mathcal{L})$ where $G_F^{(i)}$ is the graph G_F obtained after the first i blocks are processed. We define $H^{(i)}$ similarly. We assume that Cons_i is true for some i. We now bound $\Pr[\mathsf{Cons}_{i+1} | \mathsf{Cons}_i]$. We look at the step $i+1$ in H and there are three possibilities on how the edge for message block $x^{(i+1)}$ looks:

- *Fresh*: It arrives at a new vertex which is not present in $H^{(i)}$.
- *Determined*: It follows an existing edge, i.e there exists a label for the edge of the form $(x^{(i+1)}, .)$
- *Accident*: It causes an accident. In this case, $G_F^{(i+1)}$ will only be consistent if the edge corresponding to $x^{(i+1)}$ lands on the same vertex as in $H^{(i+1)}$. Note that this accident is a fresh-evaluation, i.e the output is not determined in the first i steps and is therefore chosen randomly from 2^n values. In other words, $\Pr[\mathsf{Cons}_{i+1} | \mathsf{Cons}_i] = \frac{1}{2^n}$ in this case.

Note that the third case happens $\mathrm{Acc}(H)$ times. Therefore, we have:

$$\Pr[G_F(\overline{x}, \overline{x}') = H] = \Pr[\mathsf{Cons}_{\tilde{\ell}}] \leq \frac{1}{2^{n \cdot \mathrm{Acc}(H)}} .$$

The latter inequality arises by upper-bounding $\Pr[\mathsf{Cons}_{i+1} | \mathsf{Cons}_i]$ with 1 in the case of Fresh and Determined edges. □

Lemma 5.16. *Let F be sampled randomly consistent with L, and $\overline{x} \neq \overline{x}'$. Then,*

$$\Pr[\mathrm{Acc}(G_F) \geq 2] \leq \frac{64\ell^4}{2^{2n}} + \frac{16\ell^2 q^2}{2^{2n}} .$$

Proof. We define $\mathcal{G}_L^a$, as before, to be the set of all structure graphs containing exactly a accidents.

$$\begin{aligned}
\Pr[\mathrm{Acc}(G_F) \geq 2] &= \sum_{a=2}^{\infty} \Pr[\mathrm{Acc}(G_F) \geq 2] \\
&= \sum_{a=2}^{\infty} \sum_{g \in \mathcal{G}_L^a} \Pr[G_F = g] \\
&\leq \sum_{a=2}^{\infty} \frac{|\mathcal{G}_L^a|}{2^{n \cdot a}} .
\end{aligned}$$

The last step follows from Lemma 5.15. Observe that for fixed base graph and fixed messages, entire graph is defined by list of accidents. Each accident is defined by an edge (i, j), where there are $\tilde{\ell} := \ell + \ell'$ choices for i and $\tilde{\ell} + q$ for j. Thus, there are at most $(\tilde{\ell}(\tilde{\ell} + q))^c$ graphs with a accidents.

$$\begin{aligned}
\Pr[\mathrm{Acc}(G_F) \geq 2] &\leq \sum_{a=2}^{\infty} \frac{|\mathcal{G}_L^a|}{2^{n \cdot a}} \\
&\leq \sum_{a=2}^{\infty} \left(\frac{\tilde{\ell}(\tilde{\ell} + q)}{2^n} \right)^a \\
&\leq \left(\frac{\tilde{\ell}(\tilde{\ell} + q)}{2^n} \right)^2 \leq \frac{4\tilde{\ell}^4}{2^{2n}} + \frac{4\tilde{\ell}^2 q^2}{2^{2n}} \leq \frac{64\ell^4}{2^{2n}} + \frac{16\ell^2 q^2}{2^{2n}} .
\end{aligned}$$

where we assume $\tilde{\ell} \leq 2\ell$ □

Note that $H \in \mathsf{Coll}_L = \mathsf{Coll}_L(\overline{x}, \overline{x}')$ necessarily implies $\mathsf{Acc}(H) \geq 1$. We can then say, for $G_F = G_F(\overline{x}, \overline{x}')$,

$$\begin{aligned} \Pr[G_F \in \mathsf{Coll}_L] &\leq \Pr[G_F \in \mathsf{Coll}_L \wedge \mathsf{Acc}(G_F) = 1] + \Pr[\mathsf{Acc}(G_F) \geq 2] \\ &\leq \frac{\left|\mathsf{Coll}_L \cap \mathcal{G}_L^1\right|}{2^n} + \frac{64\ell^4}{2^{2n}} + \frac{16\ell^2 q^2}{2^{2n}} \end{aligned} . \tag{5.5}$$

by Lemmas 5.15 and 5.16. We are going to upper bound the number of graphs in $\mathsf{Coll}_L \cap \mathcal{G}_L^1$.

We reduce handling the case of a general L now to the simpler case where $L = \emptyset$, so that we can resort to the counting argument of [GPR14].

Now, let $G \in \mathsf{Coll}_L \cap \mathcal{G}_L^1(\overline{x}, \overline{x}')$. Let $\mathcal{E}_B$ be the set of all blue edges which are on the $\overline{x}$- and the $\overline{x}'$-paths. Moreover, let $\mathcal{E}'_B \subseteq \mathcal{E}_B$ be the set of all *initial* blue edges, i.e., the set of blue edges on these paths which are not preceded by any red edge. We prove the following lemma.

Lemma 5.17. *If* $G \in \mathsf{Coll}_L \cap \mathcal{G}_L^1(\overline{x}, \overline{x}')$, *then* $\mathcal{E}'_B = \mathcal{E}_B$

Proof. We prove by contradiction by assuming that this is not true. In other words, there exists at least one edge $e \in \mathcal{E}_B \setminus \mathcal{E}'_B$. We now proceed to show that this necessarily leads to at least two collisions, contradicting $G \in \mathsf{Coll}_L \cap \mathcal{G}_L^1(\overline{x}, \overline{x}')$. Wlog, assume that this lies on the $\overline{x}$ path, and let $e_i = (u_i, v_i)$ be the first such edge. Then, this edge must be preceded by a red-edge, e_{i-1}. Then, clearly this implies that the graph contains at least one accident.

Now, let us take a look at the $\overline{x}'$ path. We have, by our definition of G (it is in the set Coll_L), that there should exist a path from e_i to the end point of the $\overline{x}'$ path. We will sketch a proof to show that this is not possible without having a second collision. We take a look at this path from e_i to the end point of $\overline{x}'$, V'.

- The path is of entirely blue edges.

We look at the edge into V' on the $\overline{x}'$ path. Note that this cannot be a new blue edge as it would violate the no-collision constraint on L. If this was a red edge we have a second collision, by our definition. The only option is to take the same blue edge as on the e_i-path. In other words, the penultimate vertex on e_i-path and $\overline{x}'$ path are the same. We can argue similarly for the penultimate vertex by showing that the edges into them should be the same blue edge on both the paths. Working backwards we arrive at the point u_i. Thus, we have shown that the message blocks corresponding to the edges along the path from e_i to V' is the same on both $\overline{x}$ and $\overline{x}'$. We have also assumed that e_i is the first non-initial blue edge. In other words, the path before e_i should be all red. Since $\overline{x} \neq \overline{x}'$, there should be a block before e_i which would differ in the message and hence this would constitute a collision on the red edges. Therefore, in this case it is impossible to have the final outputs to be the same without causing a second collision.

- The path is of entirely red edges.

  The argument is similar to the previous case. The only difference is that the $\overline{x}'$-path could take a blue edge into the e_i-path. However, this would still constitute a new collision. The same reasoning follows giving us the same conclusion.

- The path is mixed red and blue edges.

  We have already shown that if a blue edge follows a red edge on the path then it is a collision. We can therefore assume that the set of blue edges occur together, followed by the set of red edges. The reasoning is pretty similar to the previous two cases.

In other words, a graph G cannot be in Coll_L and $\mathcal{G}_L^1(\overline{x}, \overline{x}')$ when there exist non-initial blue edges. □

The direct consequence of Lemma 5.17 is that the number of colliding structure graphs with one accident is the same even if we remove all the non-initial blue edges. In addition, removing

the initial blue edges will also not decrease the number of colliding graphs with one accident. This is true because the initial blue edges can be replaced by red edges without increasing the number of accidents by the properties of L. In other words, if we define $L' \subseteq L$ to be the set of queries which define the initial blue edges, we have

$$\left|\mathsf{Coll}_L \cap \mathcal{G}^1_L(\overline{x}, \overline{x}')\right| = \left|\mathsf{Coll}_{L'} \cap \mathcal{G}^1_{L'}(\overline{x}, \overline{x}')\right| \leq \left|\mathsf{Coll}_{\emptyset} \cap \mathcal{G}^1_{\emptyset}(\overline{x}, \overline{x}')\right| \leq \ell d'(\ell) \ . \tag{5.6}$$

where the last inequality is from [GPR14]. Here $d'(n)$ is defined as follows:

$$d'(n) := \max_{n' \in \{1,\ldots,n\}} |\{d \in \mathbb{N} : n' \mod d = 0\}| \ .$$

Combining Equations 5.4, 5.5 and 5.6 we get,

$$\Pr[\tilde{Y}_\ell = \tilde{Y}'_\ell] \leq \frac{1}{2^r} + \frac{\ell \cdot d'(\ell)}{2^n} + \frac{64\ell^4}{2^{2n}} + \frac{16\ell^2 q^2}{2^{2n}}$$

Substituting the above value in Equations 5.2 and 5.3, we get:

$$\mathsf{SD}\left((\tilde{Y}_\ell, F)|z, (U_r, F)|z\right) \ \leq \ \frac{1}{2}\sqrt{2^r p_z + \ell \cdot d'(\ell) \cdot \frac{2^r}{2^n} + 64\ell^4 \cdot \frac{2^r}{2^{2n}} + 16\ell^2 q^2 \cdot \frac{2^r}{2^{2n}}} \ .$$

This concludes the proof of Lemma 5.11 □

5.1.3 Parameter choices

In terms of concrete parameters, observe the following for the Merkle-Damgård constructions above:

- **Computational PRNG:** If one were to use SHA-512 as compression function with $n = 512$, and, moreover, choose $r = n$. We let $t = 1, q = 2^{80}$ and let $\gamma^* = \ell$. This assumes that we get at least one bit of entropy from each block. We would need $\gamma^* \approx 160$ to get 80 bits of

security.

- **IT PRNG:** For example, assume SHA-512's compression function is used, i.e., $n = 512$. If we let $r = 256$, then we get (we also approximate $1/(1-\rho) \leq 2$, very generously)

$$\varepsilon_{\text{rob-it}} \leq \frac{t}{2}\sqrt{2^{257-\gamma^*} + \frac{\ell \cdot d'(\ell)}{2^{256}}} + t\frac{q^2}{2^{256}},$$

We let $\ell = \gamma^*$. Then, if we set for example $q = 2^{80}$. We would need the entropy loss, i.e, $\gamma^* - r = 162$ for 80 bits of security.

5.2 PRNGs from Merkle-Damgård with Davies-Meyer

The Davies-Meyer compression function maps two inputs $a \in \{0,1\}^m$ and $b \in \{0,1\}^n$ to an n-bit string

$$E(b,a) \oplus a\,,$$

where E is an arbitrary block cipher (where b is the key and a the input).[7]

5.2.1 Computational PRNGs from Merkle-Damgård with Davies-Meyer

Now, a PRNG can be obtained from E as follows (cf. Figure 5.2):

Construction 12 (PRNG from MD-DM)**.** The (n,r)-PRNG construction $\mathsf{DM} = (\mathsf{refresh}, \mathsf{next})$ based on Merkle-Damgård with Davies-Meyer (MD-DM) uses a cipher $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ and is defined as follows:[8]

- $\mathsf{refresh}^E(s,x) = E(x,s) \oplus s$, and
- $\mathsf{next}^E(s) = (E(0,s) \oplus s, E(1,s) \oplus s \| \cdots \| E(r/n,s) \oplus s)$.

[7] A (block) cipher is an efficiently computable and invertible permutation $E(k,\cdot) : \{0,1\}^n \to \{0,1\}^n$ for every key $k \in \{0,1\}^n$.

[8] The integer arguments to the cipher are to be naturally mapped to $\{0,1\}^n$.

The security of Construction 12 is proved in the E-model, where E is a cipher chosen uniformly at random from the set of all ciphers and can be queried in both the forward and backward direction.

Theorem 5.18 (Robustness of MD-DM PRNGs)**.** *Construction 12 is a* $(\gamma^*, q, t, \ell, \varepsilon_{\text{rob}})$*-robust PRNG in the E-model for*

$$\varepsilon_{\text{rob}} \;\leq\; 4t \cdot \left(\frac{\tilde{q}^2 + \tilde{q}\ell + \ell^2}{2^n} + \frac{\tilde{q}}{2^{\gamma^*}} \right),$$

where $\tilde{q} = q + r/n + 1$.

Note that iteratively absorbing some input blocks $x_1, \ldots, x_\ell$ via refresh, starting with a state s_0 is identical to applying the MD-DM construction to the input with initialization vector (IV) s_0, which is denoted by $\mathsf{MD\text{-}DM}^E_{s_0}(x_1, \ldots, x_\ell)$ in the remainder of this paper.

The robustness of the MD-DM PRNG construction is proved along similar lines as that of the MD construction with a random compression function (cf. Section 5.1.1). It is highly recommended to read that proof first. The proof again establishes extraction, maintaining, and next security, before showing that these imply recovering and preserving security. The final bound in Theorem 5.1 follows by applying Theorem 4.8.

5.2.1.1 Extraction Security

Lemma 5.19 (Extraction security)**.** *The advantage of any* γ^**-legitimate* (q, ℓ)*-attacker* $\mathcal{A}$ *against extraction security of* DM *is bounded by*

$$\mathrm{Adv}^{\text{ext},E}_{\mathsf{DM}}(\mathcal{A}) \;\leq\; \varepsilon^{\text{ext}}_{\mathsf{MD}}(\gamma^*, q, \ell) \;:=\; \frac{q^2 + 2(q\ell + \ell^2)}{2^n} + \frac{4q}{2^{\gamma^*}}\,.$$

Proof. The transcript has the identical format as previously, i.e.,

$$\tau \;=\; (s^\star, s_0, x_1, \ldots, x_\ell, L)$$

except that L is now the set of E-queries. To define bad transcripts, let $\ell' \geq 0$ be maximal such

that

$$((x_i, y_{i-1}), y_i \oplus y_{i-1}) \in L$$

for some values $y_0, y_1, \ldots, y_{\ell'}$ with $y_0 = s_0$. A transcript τ is a bad transcript if

- $\ell' = \ell$ or
- $\ell' = \ell - 1$ and there exists a query $((x_\ell, *), y_{\ell-1} \oplus s^\star) \in L$.

As before, to apply Theorem 3.1, one merely needs to bound the probability ratio for good transcripts (Lemma 5.20) and the probability of a bad transcript occurring in the ideal world, i.e, for $b = 1$ (Lemma 5.21). □

Lemma 5.20 (Ratio analysis)**.** *For all* good *transcripts* τ*,*

$$\frac{\mathrm{p}_0(\tau)}{\mathrm{p}_1(\tau)} \;\geq\; 1 - \frac{2(q\ell + \ell^2)}{2^n}\,.$$

Proof. As in Section 5.1, for a good transcript,

$$\mathrm{p}_1(\tau) \;=\; p_L \cdot 2^{-n} \quad \text{and} \quad \mathrm{p}_0(\tau) \;=\; p_L \cdot q_\tau\,,$$

where p_L denotes the probability that a uniform random cipher is consistent with the queries in L and where q_τ is the probability that

$$\mathsf{MD\text{-}DM}^{E_L}_{s_0}(x_1, x_2, \ldots, x_\ell) \;=\; s^\star$$

over a cipher E_L that is sampled uniformly at random conditioned on being consistent with L.

To derive a lower bound on q_τ, due to τ being a good transcript, it suffices to consider the two cases

(1) $\ell' \leq \ell - 2$ and

(2) $\ell' = \ell - 1$.

For (1), q_τ is the probability (over E_L) that

$$Y_\ell \;:=\; \mathsf{DM}^{E_L}_{y_{\ell'}}(x_{\ell'+1}, \ldots, x_\ell) \;=\; s^\star \;.$$

Consider the intermediate chaining values $Y_{\ell'} = y_{\ell'}, Y_{\ell'+1}, \ldots, Y_{\ell-1}$ (defined via evaluations of E). Moreover, let $L_{\ell'}, L_{\ell'+1}, \ldots, L_{\ell-2}$ be the set of points at which E is defined after evaluating these intermediate values, i.e., $L_{\ell'} := L$ and

$$L_i \;:=\; L_{i-1} \cup \{((x_i, Y_{i-1}), Y_i \oplus Y_{i-1})\}$$

for $i = \ell' + 1, \ldots, \ell - 2$. Furthermore, for a set $\tilde{L}$ of E-queries define the sets $\mathsf{Free\text{-}In}_{\tilde{L}}$ of *free inputs* and $\mathsf{Free\text{-}Out}_{\tilde{L}}$ of *free outputs*, i.e.,

$$y \in \mathsf{Free\text{-}In}_{\tilde{L}} :\Longleftrightarrow ((*, y), *) \notin \tilde{L} \quad \text{and} \quad y \in \mathsf{Free\text{-}Out}_{\tilde{L}} :\Longleftrightarrow ((*, *), y) \notin \tilde{L} \;.$$

Finally, for $i = \ell' + 1, \ldots, \ell - 2$, define the event FRESH_i that Y_i is a *fresh input*, i.e.,

$$Y_i \in \mathsf{Free\text{-}In}_{L_{i-1}} \;.$$

However, for $Y_{\ell-1}$, let $\mathsf{FRESH}_{\ell-1}$ be the event *not only* that $Y_{\ell-1} \in \mathsf{Free\text{-}In}_{L_{\ell-2}}$ but also that $Y_{\ell-1} \oplus s^\star$ is a free *output*, i.e.,

$$Y_{\ell-1} \oplus s^\star \in \mathsf{Free\text{-}Out}_{L_{\ell-2}} \;.$$

Let,

$$\mathsf{FRESH} \;:=\; \bigcap_{i=\ell'+1}^{\ell-1} \mathsf{FRESH}_i \;.$$

Then, on the one hand,

$$\begin{aligned}\Pr[Y_\ell = s^\star | \mathsf{FRESH}] &= \Pr[E(x_\ell, Y_{\ell-1}) = Y_{\ell-1} \oplus s^\star | \mathsf{FRESH}] \\ &= \frac{1}{|\mathsf{Free\text{-}Out}_{L_{\ell-2}}|} \geq \frac{1}{2^n}\end{aligned}$$

since the conditioning implies that $(x_\ell, Y_{\ell-1})$ is a fresh input to E and that $Y_{\ell-1} \oplus s^\star$ actually is in $\mathsf{Free\text{-}Out}_{L_{\ell-2}}$. On the other hand, in order to bound

$$\Pr\left[\overline{\mathsf{FRESH}_{\ell-1}} \,\middle|\, \bigcap_{k=\ell'+1}^{\ell-2} \mathsf{FRESH}_k\right],$$

observe that, for any values $y_{\ell'+1}, \ldots, y_{\ell-2}$ consistent with the conditioning, $\mathsf{FRESH}_{\ell-1}$ is violated if[9]

$$E_L(x_{\ell-1}, y_{\ell-2}) \in \overline{\mathsf{Free\text{-}In}_{L_{\ell-2}}} \oplus y_{\ell-2}$$

or

$$E_L(x_{\ell-1}, y_{\ell-2}) \in \overline{\mathsf{Free\text{-}Out}_{L_{\ell-2}}} \oplus y_{\ell-2} \oplus s^\star \,.$$

The probability that the former condition is violated is at most

$$\begin{aligned}\frac{|\mathsf{Free\text{-}Out}_{L_{\ell-2}} \cap \overline{\mathsf{Free\text{-}In}_{L_{\ell-2}}} \oplus y_{\ell-2}|}{|\mathsf{Free\text{-}Out}_{L_{\ell-2}}|} &\leq \frac{|\overline{\mathsf{Free\text{-}In}_{L_{\ell-2}}} \oplus y_{\ell-2}|}{|\mathsf{Free\text{-}Out}_{L_{\ell-2}}|} \\ &\leq \frac{q+\ell}{2^n - (q+\ell)} \,.\end{aligned}$$

The same bound via a similar argument is obtained for the latter condition as well as

$$\Pr\left[\overline{\mathsf{FRESH}_i} \,\middle|\, \bigcap_{k=\ell'+1}^{i-1} \mathsf{FRESH}_k\right] \leq \frac{q+\ell}{2^n - (q+\ell)}$$

[9] Using the notation $A \oplus b = \{x \oplus b \mid x \in A\}$.

for $i = \ell' + 1, \ldots, \ell - 2$. Therefore,

$$\Pr[\mathsf{FRESH}] \;\geq\; 1 - \frac{q\ell + \ell^2}{2^n - (q+\ell)}\,,$$

and, finally,

$$\begin{aligned} q_\tau \;&\geq\; \Pr[\mathsf{FRESH}] \cdot \Pr[Y_\ell = s^\star | \mathsf{FRESH}] \\ &\geq\; \left(1 - \frac{q\ell + \ell^2}{2^n - (q+\ell)}\right) \cdot 2^{-n}\,, \end{aligned}$$

for case (1). For case (2), note that due to $\ell' = \ell - 1$, $(x_\ell, y_{\ell-1})$ is a fresh input to E, and, furthermore, $y_{\ell-1} \oplus s^\star \in \mathsf{Free\text{-}Out}_L$. Thus, in this case the probability that $Y_\ell = s^\star$ is at least 2^{-n}.

Combining both cases, case (1) dominating, one obtains

$$\frac{\mathrm{p}_0(\tau)}{\mathrm{p}_1(\tau)} \;\geq\; 1 - \frac{q\ell + \ell^2}{2^n - (q+\ell)} \;\geq\; 1 - \frac{2(q\ell + \ell^2)}{2^n}\,,$$

using that $q + \ell \leq 2^{n-1}$, an assumption one may always make since the bound in the lemma is vacuous otherwise. □

Lemma 5.21 (Bad event analysis)**.** *For the set $\mathcal{B}$ of bad transcripts (as defined above),*

$$\Pr[T_1 \in \mathcal{B}] \;\leq\; \frac{4q}{2^{\gamma^*}} + \frac{q^2}{2^n}\,.$$

Proof. The same resampling approach as in the proof of Lemma 5.4 applies here as well. However, the collisions require additional care. Two queries E queries $((k,u),v)$ and $((k',u'),v')$ are said to *collide* if

$$v \oplus u \;=\; v' \oplus u'\,.$$

It is easily verified that the probability, over E, that any two such queries collide is at most $(2^n -$

$1)^{-1}$. Let $\mathcal{E}$ be the event that there exists such a collision in $\mathcal{L}$. Note that

$$\Pr[T_1 \in \mathcal{B}] \;\leq\; \Pr[T_1 \in \mathcal{B}|\mathcal{E}] + \Pr[\overline{\mathcal{E}}] \;\leq\; \Pr[T_1 \in \mathcal{B}|\mathcal{E}] + \frac{q^2}{2^n}\,.$$

Towards bounding $\Pr[T_1 \in \mathcal{B}|\mathcal{E}]$, consider a triple $z = (\sigma, L, s_0) \in \mathcal{E}$, which, once more, is shorthand for L being collision-free. Define a *potential chain* as ℓ, for *some* ℓ, values $y_0, y_1, \ldots, y_\ell$ such that $y_0 = s_0$ and, for some values $k_1, \ldots, k_\ell$,

(a) $((k_i, y_{i-1}), y_i \oplus y_{i-1}) \in L$ for $i = 1, \ldots, \ell$ or

(b) $((k_i, y_{i-1}), y_i \oplus y_{i-1}) \in L$ for $i = 1, \ldots, \ell - 1$ and $((k_\ell, y_\ell), y_{\ell-1} \oplus s^\star) \in L$.[10]

Without collisions, L can contain at most $2q$ potential chains; this can be proved by induction: Consider a set collision-free set L' of E-queries and assume that the number of potential chains is at most $2|L'|$. Consider an additional query $((k, u), v)$ that does not cause a collision; it may only (but need not) create a new potential chain in two ways:

- $u = v' \oplus u'$: this corresponds to extending in the sense of (a) above and can only be true for a single previous query $((k', u'), v')$ if L is collision-free;
- $v = v' \oplus u' \oplus s^\star$: this corresponds to creating a chain of type (b) and, again, can only hold for one query $((k', u'), v') \in$ if L is collision-free.

Summarizing, the new query creates at most two new potential chains.

Clearly, conditioned on $Z = z$, $T_1 \in \mathcal{B}$ if and only if for some potential chain, $X_i = k_i$ for all $i = 1, \ldots, \ell$. Hence, by the legitimacy of $\mathcal{A}$,

$$\Pr[T_1 \in \mathcal{B}|Z = z] \;\leq\; 2q \cdot p_z\,,$$

[10] Observe that in case (b) the "intuitive chain" goes up to $y_{\ell-1}$ but y_ℓ is not part of it.

where $p_z := \mathsf{Pred}(\overline{X}|Z = z)$. The remainder of the proof proceeds in the exact same fashion as the proof of Lemma 5.4. □

5.2.1.2 Maintaining Security

The maintaining security of the Davies-Meyer PRNG construction DM is proved along similar lines as that of the MD construction with a random compression function. This section discusses the few differences.

Lemma 5.22 (Maintaining security)**.** *The advantage of any (q, ℓ)-attacker $\mathcal{A}$ against maintaining security is bounded by*

$$\mathrm{Adv}^{\mathrm{mtn},E}_{\mathsf{DM}}(\mathcal{A}) \;\leq\; \varepsilon^{\mathrm{mtn}}_{\mathsf{DM}}(q,\ell) \;:=\; \frac{2(q\ell+\ell^2)}{2^n} + \frac{q}{2^n}\,.$$

Proof. Similarly to Lemma 5.5, maintaining security is shown via an H-coefficient proof. Once more, one considers transcripts

$$\tau = (s^\star, s_0, x_1, \ldots, x_\ell, L)\,,$$

with the difference that L refers to E-queries here. A *bad* transcript contains a query of the type $((*, s_0), *) \in L$. Again, the probability of a bad transcript occurring in the $b = 1$ case is at most $|L|/2^n$, and for good transcripts,

$$\mathrm{p}_1(\tau) \;=\; 2^{-2n} \cdot p_L \quad \text{and} \quad \mathrm{p}_0(\tau) \;\geq\; \left(1 - \frac{2(q\ell+\ell^2)}{2^n}\right) \cdot 2^{-2n} \cdot p_L\,,$$

where the latter follows via an argument similar to that in the proof of Lemma 5.20. □

5.2.1.3 Next Security

Next security of DM is defined and proved in a fashion analogous to the case with a random compression function.

Lemma 5.23 (Next security)**.** *The advantage of any q-attacker $\mathcal{A}$ against next security is bounded by*

$$\mathrm{Adv}^{\mathrm{next},E}_{\mathsf{DM}}(\mathcal{A}) \;\leq\; \varepsilon^{\mathrm{next}}_{\mathsf{DM}}(q) \;:=\; \frac{q}{2^n}\,.$$

Proof. For a straight-forward H-coefficient proof, consider the transcript

$$\tau = (s, s_0^\star, s_1^\star, \ldots, s_{r/n}^\star, L)\,,$$

where s is the initial state, the values $s_i^\star$ are the input to $\mathcal{A}$, and L are the queries $\mathcal{A}$ makes to F. A transcript is *bad* if L contains a query of the form $((i,s),*)$ or $((i,*), s \oplus s_i^\star)$ for some i; otherwise, τ is called *good*. It is easily seen that for good transcripts, the probability ratio is *at least* 1, and, in the ideal world, where $\mathcal{A}$'s view is completely independent of S, the probability of a bad event is at most $q/2^n$. □

5.2.1.4 Recovering Security

In the following, let $\varepsilon^{\mathrm{ext}}_{\mathsf{DM}}(\gamma^*, q, \ell)$ and $\varepsilon^{\mathrm{next}}_{\mathsf{DM}}(q)$ be as in Lemmas 5.19 and 5.23. Once more, extraction and next security together imply recovering security:

Lemma 5.24 (Recovering Security)**.** *For every γ^*-legitimate (q,ℓ)-attacker $\mathcal{A}$,*

$$\mathrm{Adv}^{\mathrm{rec},E}_{\mathsf{DM}}(\mathcal{A}) \;\leq\; \varepsilon^{\mathrm{ext}}_{\mathsf{DM}}(\gamma^*, \ell, q + r/n + 1) + \varepsilon^{\mathrm{next}}_{\mathsf{DM}}(q + r/n + 1)\,.$$

The proof of the lemma is completely analogous to that of Lemma 5.7 and is omitted.

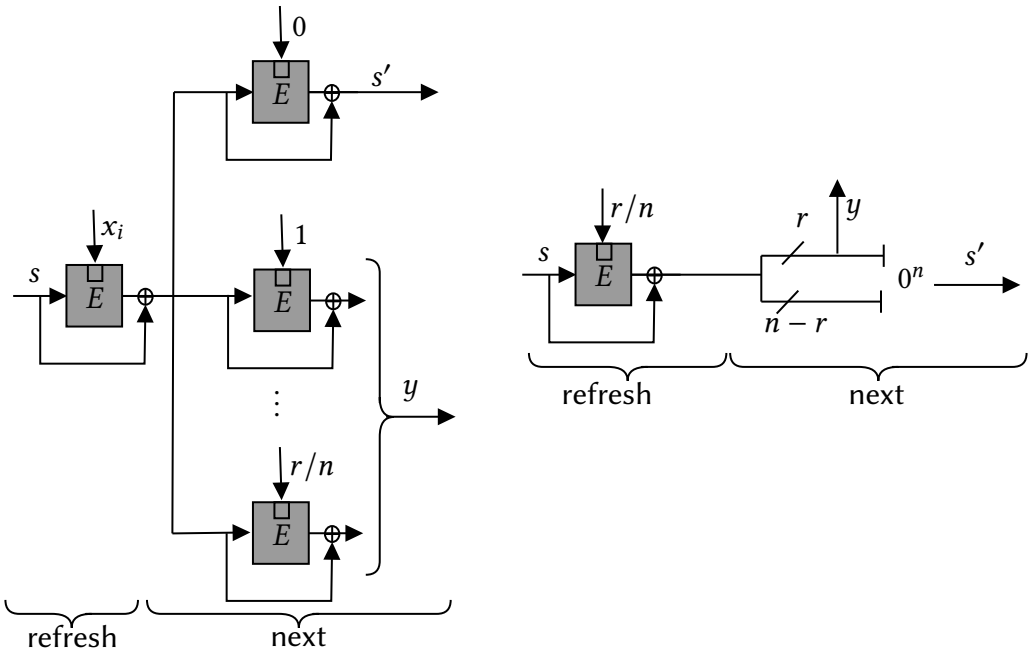

Figure 5.2: Procedures refresh (processing a single-block input x_i) and next of Merkle-Damgård PRNG constructions with the Davies-Meyer compression function based on a block cipher E. Left: Computationally secure Construction 12; right: IT secure Construction 13.

5.2.1.5 Preserving Security

In the following, let $\varepsilon^{\text{mtn}}_{\text{MD}}(q,\ell)$ and $\varepsilon^{\text{next}}_{\text{MD}}(q)$ be as in Lemmas 5.5 and 5.6. Maintaining and next security together imply preserving security:

Lemma 5.25 (Preserving Security)**.** *For every adversary $\mathcal{A}$,*

$$\text{Adv}^{\text{pre},E}_{\text{DM}}(\mathcal{A}) \;\leq\; \varepsilon^{\text{mtn}}_{\text{DM}}(q + r/n + 1, \ell) + \varepsilon^{\text{next}}_{\text{DM}}(q + r/n + 1)\ .$$

The proof of the lemma is completely analogous to that of Lemma 5.8 and is omitted.

5.2.2 IT PRNGs from Merkle-Damgård with Davies-Meyer

In the IT-secure variant of the MD-DM construction, refresh remains the same, but next will truncate the input state to r bits, which it outputs, and then zero out the state.

Construction 13 (IT-PRNG from MD-DM)**.** The (n, r)-PRNG construction $\mathsf{DM}_r = (\mathsf{refresh}, \mathsf{next})$ using Merkle-Damgård with Davies-Meyer (MD-DM) uses a block cipher $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ and is defined as follows:

- $\mathsf{refresh}^E(s, x) = E(x, s) \oplus s$, and
- $\mathsf{next}^E(s) = (0^n, s[1..r])$.

The security of Construction 13 is proved in the E-model, where E is a cipher chosen uniformly at random from the set of all ciphers and can be queried in both the forward and backward direction. Let $d'(\ell)$ be defined as in Section 5.1.

Theorem 5.26 (IT-Robustness of MD-DM PRNGs)**.** *Construction 13 is a $(\gamma^*, q, t, \ell, \varepsilon_{\text{rob}})$-IT-robust PRNG in the E-model, where*

$$\varepsilon_{\text{rob-it}} \leq \frac{t}{2}\sqrt{\frac{2^{r-\gamma^*}}{(1-\rho)} + \ell \cdot d'(\ell)\frac{2^r}{2^{n-1}} + 64\ell^4 \cdot \frac{2^r}{2^{2n-2}} + 16\ell^2\tilde{q}^2 \cdot \frac{2^r}{2^{2n-2}}} + t\rho\,,$$

for $\rho = \frac{\tilde{q}^2}{2^r}$ where $\tilde{q} = q + t\ell$

Our robustness proof for this construction uses the analysis from Section 5.1.2. The intuition here is that the structure graphs were defined for a compressing function F and we instantiate it with a Davies-Meyer construction and extend the arguments. However, there is a subtle difference because of the underlying primitive. In Davies-Meyer the primitive is an ideal cipher. Thus, we state and prove Lemma 5.29 which is the counterpart of Lemma 5.15 for the Davies-Meyer instantiation. We then apply the results of Lemma 5.29 in the proofs of Lemma 5.16 and results from [GPR14] to prove Theorem 5.18.

Proof. The proof follows from Lemma 5.27 and Theorem 4.12. □

Lemma 5.27. *For every γ^*-IT-legitimate (q, ℓ)-attacker, in the ideal cipher model,*

$$\mathrm{Adv}^{\text{rec-IT},\gamma^*}_{\mathsf{DM}}(\mathcal{A}) \leq \frac{1}{2}\sqrt{\frac{2^{r-\gamma^*}}{(1-\rho)} + \ell \cdot d'(\ell)\frac{2^r}{2^{n-1}} + \frac{64\ell^4 2^r}{2^{2n-2}} + \frac{16\ell^2 q^2 2^r}{2^{2n-2}}} + \rho \ .$$

where $\rho = \frac{q^2}{2^r}$

Proof. The arguments for this proof is similar to the proof of Lemma 5.10. As before we define the random variables $\tilde{Y}_\ell, Z, \Sigma, U_r$. The difference arises in the definition of $Z = (\Sigma, \mathcal{L}, S_0)$. We define the $\mathcal{L}$ as follows: If $\mathcal{L}'$ is the set of queries made by $\mathcal{A}$ to the ideal cipher E, then for every $((x, y), z) \in \mathcal{L}'$, add $((x, y), z \oplus y)$ to $\mathcal{L}$. This is to ensure that the $\mathcal{L}$ is consistent with the evaluation of Davies-Meyer on the input (x, y). We now upper-bound the adversary $\mathcal{A}$ in the IT-recovering game by showing an upperbound for $\mathsf{SD}\left((\tilde{Y}_\ell, Z, E), (U_r, Z, E)\right)$.

We also define the event $\mathcal{E}$ which denotes the event when $\mathcal{L}$ entries are distinct *when truncated to the first r bits*. Note that there are a maximum of q queries in this list. We would like to point out that $\mathcal{E}$ has the same probability of occurring in either experiment. Therefore, by Proposition 3.4,

$$\mathsf{SD}\left((\tilde{Y}_\ell, Z, E), (U_r, Z, E)\right) \leq \mathsf{SD}\left((\tilde{Y}_\ell, Z, E)|\mathcal{E}, (U_r, Z, E)|\mathcal{E}\right) + \frac{q^2}{2^r} \ ;$$

We let $\rho := q^2/2^r$ for the remainder of the proof. In order to bound the statistical distance conditioned on $\mathcal{E}$, we can rewrite the same as

$$\mathsf{SD}\left((\tilde{Y}_\ell, Z, E)|\mathcal{E}, (U_r, Z, E)|\mathcal{E}\right) = \sum_{z \in \mathcal{E}} \Pr[Z = z|\mathcal{E}] \cdot \mathsf{SD}\left((\tilde{Y}_\ell, E)|z, (U_r, E)|z\right) \ , \tag{5.7}$$

where $z \in \mathcal{E}$ is to denote that the sum is taken over all side informations $Z = z$, satisfying $\mathcal{E}$.[11]

[11] Therefore $\mathcal{E}$ can be omitted in the conditioning of the statistical distance.

We define $p_z := \mathsf{Pred}(\overline{X}|Z = z)$, and as shown in Lemma 5.10, we have

$$\mathrm{H}_\infty(\overline{X}|Z\mathcal{E}) \;\geq\; \gamma^* - \log(1-\rho)^{-1}\,,$$

From Lemma 5.28 we get,

$$\mathsf{SD}\left((\tilde{Y}_\ell, E)|z, (U_r, E)|z\right) \leq \frac{1}{2}\sqrt{2^r p_z + \ell \cdot d'(\ell)\frac{2^r}{2^{n-1}} + 64\ell^4\frac{2^r}{2^{2n-2}} + 16\ell^2 q^2\frac{2^r}{2^{2n-2}}}$$

Using Jensen's inequality, (5.7) becomes, for $\alpha = 2^r$ and $\beta = \ell \cdot d'(\ell) \cdot \frac{2^r}{2^{n-1}} + 64\ell^4 \cdot \frac{2^r}{2^{2n-2}} + 16\ell^2 q^2 \cdot \frac{2^r}{2^{2n-2}}$,

$$\begin{aligned}\mathsf{SD}\left((\tilde{Y}_\ell, E)|\mathcal{E}, (U_r, E)|\mathcal{E}\right) &\leq \frac{1}{2}\sqrt{\alpha\mathsf{Pred}(\overline{X}|\mathcal{L}\mathcal{E}) + \beta} \\ &\leq \frac{1}{2}\sqrt{\frac{2^{r-\gamma^*}}{(1-\rho)} + \ell \cdot d'(\ell)\frac{2^r}{2^{n-1}} + 64\ell^4\frac{2^r}{2^{2n-2}} + 16\ell^2\frac{q^2 2^r}{2^{2n-2}}}\,.\end{aligned}$$

□

Lemma 5.28. *For $z \in \mathcal{E}$ and the random variables as defined earlier,*

$$\mathsf{SD}\left((\tilde{Y}_\ell, E)|z, (U_r, E)|z\right) \leq \frac{1}{2}\sqrt{2^r p_z + \ell \cdot d'(\ell)\frac{2^r}{2^{n-1}} + 64\ell^4\frac{2^r}{2^{2n-2}} + 16\ell^2 q^2\frac{2^r}{2^{2n-2}}}$$

Proof. We fix $z = (\sigma, L, s_0)$. We have that z, E is distributed uniformly over the set of all ideal ciphers that agree with L. Thus, by Proposition 3.3,

$$\mathsf{SD}\left((\tilde{Y}_\ell, E)|z, (U_r, E)|z\right) \;\leq\; \frac{1}{2}\sqrt{2^r \cdot \mathsf{Coll}(\tilde{Y}_\ell|Ez) - 1}\,. \tag{5.8}$$

To bound the collision probability, we consider the following experiment:

- choose E uniformly consistent with L
- sample inputs $\overline{X} = (\overline{X}_1, \ldots, \overline{X}_\ell)$ and $\overline{X}' = (\overline{X}'_1, \ldots, \overline{X}'_{\ell'})$ independently but conditioned on

$Z = z$.

- compute $\tilde{Y}_\ell$ and $\tilde{Y}'_\ell$ as the truncated MD evaluations with E of $\overline{X}$ and $\overline{X}'$

Now, we have that:

$$\Pr[\tilde{Y}_\ell = \tilde{Y}'_\ell] \;\leq\; \Pr[\overline{X} = \overline{X}'] + \Pr[\tilde{Y}_\ell = \tilde{Y}'_\ell | \overline{X} \neq \overline{X}'] \; . \tag{5.9}$$

Clearly, the former is at most p_z. To get a bound on the second term, we proceed to fix arbitrary inputs $\overline{x} \neq \overline{x}'$ of lengths ℓ and ℓ', respectively. This is similar to the process that is adopted in the proof of Lemma 5.11. We construct a similar Structure Graph. Note that in Lemma 5.11, we assumed that the primitive was a compressing function. In particular, Davies Meyer is also a compressing function. Therefore, we can define similar structure graph for this construction by viewing $F(y, x) := E(x, y) \oplus y$. From Lemma 5.14 we get the following result:

$$\Pr[\tilde{Y}_\ell = \tilde{Y}'_\ell] \leq \Pr[G_F(\overline{x}, \overline{x}') \in \mathsf{Coll}_L(\overline{x}, \overline{x}')] + \frac{1}{2^r} \; .$$

However, the proof of Lemma 5.15 changes. We prove a corresponding Lemma as follows:

Lemma 5.29. *For an E sampled randomly consistent with L with F defined as above, and $\overline{x} \neq \overline{x}'$. Let $H \in \mathcal{G}_L(\overline{x}, \overline{x}')$. Then,*

$$\Pr[G_F(\overline{x}, \overline{x}') = H] = \frac{1}{2^{(n-1)\cdot \mathsf{Acc}(H)}} \; .$$

Proof. We have two messages $\overline{x} \neq \overline{x}'$ and let $x^{(i)}$ be as defined before. We have a randomly sampled E which is consistent with L. The proof is similar to Lemma 5.15. However, the proof differs in the case of a colliding edge, i.e the edge causes an accident. In this case, $G_F^{(i+1)}$ will only be consistent if the edge corresponding to $x^{(i+1)}$ lands on the same vertex as in $H^{(i+1)}$. However, the evaluation is not a random function F which chooses a value, uniformly at random, from 2^n

values. The evaluation is dependent on the definition of the ideal cipher E. The output in this case is chosen, from a minimum of $2^n - q - i$ values. In other words, $\Pr[\mathsf{Cons}_{i+1}|\mathsf{Cons}_i] \leq \frac{1}{2^n-q-i} \leq \frac{1}{2^{(n-1)}}$ in this case. Note that the third case happens $\mathrm{Acc}(H)$ times. Therefore, we have:

$$\Pr[G_F(\overline{x}, \overline{x}') = H] = \Pr[\mathsf{Cons}_{\tilde{\ell}}] \leq \frac{1}{2^{(n-1)\cdot \mathrm{Acc}(H)}} .$$

□

Applying this Lemma to the proof of Lemma 5.16 and the result from [GPR14], we get:

$$\mathrm{SD}\left((\tilde{Y}_\ell, E)|z, (U_r, E)|z\right) \leq \frac{1}{2}\sqrt{2^r p_z + \ell \cdot d'(\ell)\frac{2^r}{2^{n-1}} + 64\ell^4\frac{2^r}{2^{2n-2}} + 16\ell^2 q^2 \frac{2^r}{2^{2n-2}}} .$$

□

5.2.3 Parameter Choices

In terms of concrete parameters, observe the following for the PRNG constructions from Merkle-Damgård with Davies-Meyer above:

- **Computational PRNG:** SHA-512 is a 512-bit block cipher algorithm that encrypts 512 bit hash value using the input as key. Therefore, we let $n = 512$ and set $r = n$. We let $t = 1, q = 2^{80}$ and let $\ell = \gamma^*$. This assumes that we get at least one bit of entropy from each block. We would need $\gamma^* \approx 163$ to get 80 bits of security.
- **IT PRNG:** We again let $n = 512$. If we let $r = 256$, then we get (we also approximate $1/(1-\rho) \leq 2$, very generously)

$$\varepsilon_{\text{rob-it}} \leq \frac{t}{2}\sqrt{2^{129-\gamma^*} + \frac{\ell \cdot d'(\ell)}{2^{127}}} + t\frac{q^2}{2^{128}} ,$$

We let $\ell = \gamma^*$. Then, if we set for example $q = 2^{80}$. We would need the entropy loss, i.e, $\gamma^* - r = 162$ for 80 bits of security.

5.3 PRNGs from Sponges

Let $n \in \mathbb{N}$ and $n = r+c$. In the following, for an n-bit string s, let $s = s^{(r)}||s^{(c)}$ be decomposition of s into an r-bit and c-bit string.

5.3.1 Computational PRNGs from Sponges

A PRNG using the Sponge paradigm can be obtained from a permutation π as follows (cf. Figure 5.3):

Construction 14 (PRNG from Sponges)**.** The Sponge-based PRNG construction Spg = (refresh, next) uses a permutation $\pi : \{0,1\}^n \to \{0,1\}^n$ to absorb and produce r-bit inputs and outputs, respectively, and is defined as follows:

- $\mathsf{refresh}^\pi(s, x) = \pi(s \oplus x||0^c)$, and
- $\mathsf{next}^\pi(s) = (\pi(s) \oplus 0^r||s^{(c)}, s^{(r)})$.

The next function design is due to Hutchinson [Hut16a], who simplified a proposal by Gazi and Tessaro [GT16]. Recall that the Merkle-Damgård constructions have a "parallel" next function in order to produce r/n blocks of random output with $r/n + 1$ calls to the ideal primitive, where the additional call is used to produce a new state. Were it not for this optimization, on order to obtain r bits of output, one would have to apply the next function r/n times in a row, which would results in twice the number of ideal-primitive calls.

The next function for Sponges, on the other hand, only makes a single call to the ideal primitive to produce both a new state and the random output. Therefore, no parallel next function is provided for the Sponge-based PRNG.

The security of Construction 14 is proved in the π-model, where π is a uniformly random permutation, which can be queried in both the forward and backward direction.

Theorem 5.30 (Robustness of Sponge PRNGs)**.** *Construction 14 is a* $(\gamma^*, q, t, \ell, \varepsilon_{\text{rob}})$*-robust PRNG in the* π*-model for*

$$\varepsilon_{\text{rob}} \leq 4t \cdot \left(\frac{\tilde{q}^2 + \tilde{q}\ell + \ell^2}{2^n} + \frac{\tilde{q}}{2^{\gamma^*}} + \frac{\tilde{q}^2}{2^c} \right),$$

where $\tilde{q} = q + r/n + 1$.

Remark 2. Observe that the bound in Theorem 5.30 is only reasonable when c is large enough, which matches the fact that CBC-based PRNGs—which correspond to the case $c = 0$, are not secure.

Note that iteratively absorbing some input blocks $x_1, \ldots, x_\ell$ via refresh, starting with a state s_0 is identical to applying the Sponge construction to the input with initialization vector (IV) s_0, which is denoted by $\text{Sponge}^{\pi}_{s_0}(x_1, \ldots, x_\ell)$ in the remainder of this section.

The proof of the robustness of the Sponge PRNG construction follows the same outline as that of the MD construction (cf. Section 5.1.1). It is highly recommended to read that proof first. One crucial difference between the Merkle-Damgård constructions and Sponges is that Sponges do *not* satisfy extraction/maintaining security (with good parameters). For example, given the state of a Sponge PRNG after absorbing a single (possibly high-entropy) input, a simple inverse query to π results in a value of the form $a \| 0^c$, which is unlikely to happen in the ideal world ($b = 1$). This is handled by explicitly introducing a "hit" probability, i.e., the probability that the attacker queries π^{-1} on the final state of the Sponge. Recovering and preserving security are then established by arguing that the hit probability is low when next is applied to the state. The final bound in Theorem 5.30 follows by applying Theorem 4.8.

5.3.1.1 Extraction Security

The extraction security of the Sponge PRNG construction is defined and proved along similar lines as that of the previous constructions. This section discusses the differences. Denote by $\mathrm{Adv}^{\mathrm{inv},\pi}_{\mathrm{Spg}}(\mathcal{A})$ the probability that $\mathcal{A}$ queries π^{-1} on the value $s^\star$ returned by the challenger.

Lemma 5.31 (Extraction security)**.** *The advantage of any γ^*-legitimate (q,ℓ)-attacker $\mathcal{A}$ against extraction security of* DM *is bounded by*

$$\mathrm{Adv}^{\mathrm{ext},\pi}_{\mathrm{Spg}}(\mathcal{A}) \;\le\; \varepsilon^{\mathrm{ext}}_{\mathrm{Spg}}(\gamma^*,\ell,q) \;:=\; \frac{q+2(q\ell+\ell^2)}{2^n} + \frac{2q}{2^{\gamma^*}} + \frac{q^2}{2^c} + \mathrm{Adv}^{\mathrm{inv},\pi}_{\mathrm{Spg}}(\mathcal{A})\ .$$

In the following, for convenience, let

$$\delta^{\mathrm{ext}}_{\mathrm{Spg}}(\gamma^*,\ell,q) \;:=\; \frac{q+2(q\ell+\ell^2)}{2^n} + \frac{2q}{2^{\gamma^*}} + \frac{q^2}{2^c}\ .$$

Proof. The transcript has the identical format as previously, i.e.,

$$\tau \;=\; (s^\star, s_0, x_1, \ldots, x_\ell, L)$$

except that L is now the set of π-queries. To define bad transcripts let $\ell' \ge 0$ be maximal such that there exist $(u_i, v_i) \in L$ with $u_1 = s_0 \oplus x_1 \| 0^c$ and

$$u_i = v_{i-1} \oplus x_i \| 0^c$$

for $i = 2, \ldots, \ell - 1$. A transcript τ is a bad transcript if

- **(hit)** $(*, s^\star) \in L$ or
- **(chain)** $\ell' = \ell$.

As before, to apply Theorem 3.1, one merely needs to bound the probability ratio for good transcripts (Lemma 5.32) and the probability of a bad transcript occurring in the ideal world, i.e, for $b = 1$ (Lemma 5.33). □

Lemma 5.32 (Ratio analysis)**.** *For all* good *transcripts* τ*,*

$$\frac{\mathrm{p}_0(\tau)}{\mathrm{p}_1(\tau)} \;\geq\; \left(1 - \frac{2(q\ell + \ell^2)}{2^n}\right) .$$

Proof. As in Section 5.1, for a good transcript,

$$\mathrm{p}_1(\tau) \;=\; p_L \cdot 2^{-n} \quad \text{and} \quad \mathrm{p}_0(\tau) \;=\; p_L \cdot q_\tau \,,$$

where p_L denotes the probability that a uniform random permutation is consistent with the queries in L and where q_τ is the probability that

$$\mathsf{Sponge}^{\pi_L}_{s_0}(x_1, x_2, \ldots, x_\ell) \;=\; s^\star$$

over a permutation π_L that is sampled uniformly at random conditioned on being consistent with L.

To derive a lower bound on q_τ, note that due to τ being a good transcript $\ell' < \ell$. Hence, q_τ is the probability (over π_L) that

$$V_\ell \;:=\; \mathsf{Sponge}^{\pi_L}_{v_{\ell'}}(x_{\ell'+1}, \ldots, x_\ell) \;=\; s^\star \,.$$

Consider the intermediate values $U_{\ell'+1}, V_{\ell'+1}, U_{\ell'+2}, V_{\ell'+2}, \ldots, V_{\ell-1}, U_\ell$, where

$$U_{\ell'+1} \;=\; v_{\ell'} \oplus x_{\ell'+1} \| 0^c$$

and

$$U_i = V_{i-1} \oplus x_i \| 0^c$$

for $i = \ell' + 2, \ldots, \ell$ as well as

$$V_i = \pi(U_i)$$

for $i = \ell' + 1, \ldots, \ell - 1$. Define, for $i = \ell' + 1, \ldots, \ell$, the event $\mathsf{FRESH}_\mathsf{i}$ that

- U_i is *fresh*, i.e., there is no query of type $(U_i, *) \in L$ and $U_i \neq U_j$ for $j < i$, and
- V_{i-1} is not a *hit*, i.e., $V_i \neq s^\star$;

observe that $\mathsf{FRESH}_{\ell'+1}$ is always true due to the maximality of ℓ' and the fact that τ is a good transcript. Let,

$$\mathsf{FRESH} \;:=\; \bigcap_{i=\ell'+1}^{\ell} \mathsf{FRESH}_\mathsf{i} \,.$$

Then,

$$\Pr[V_\ell = s^\star | \mathsf{FRESH}] \;\geq\; 2^{-n}$$

since the conditioning implies that U_ℓ is a fresh input and therefore V_ℓ is chosen uniformly form a set of size at most 2^n, which contains $s^\star$ due to τ being a good transcript and no hits occurring while evaluating the intermediate values.

Moreover, observe that if U_{i-1} is a fresh input, the probability that V_{i-1} hits is at most $(2^n - (q+\ell))^{-1}$, and the probability that U_i is not fresh is at most $(q+\ell)(2^n - (q-\ell))^{-1}$ as there are at most $q + \ell$ non-fresh values when U_i is sampled uniformly from a set of size at least $2^n - (q + \ell)$. Hence,

$$\Pr\left[\overline{\mathsf{FRESH}_\mathsf{i}} \;\middle|\; \bigcap_{k=\ell'+1}^{i-1} \mathsf{FRESH}_\mathsf{k}\right] \;\leq\; \frac{q+\ell+1}{2^n - (q+\ell)},$$

and

$$\Pr[\mathsf{FRESH}] \;\geq\; 1 - \frac{q\ell + \ell^2 + \ell}{2^n - (q-\ell)} \,,$$

and, finally,

$$\begin{aligned} q_\tau &\geq \Pr[\mathsf{FRESH}] \cdot \Pr[Y_\ell = s^\star | \mathsf{FRESH}] \\ &\geq \left(1 - \frac{q\ell + \ell^2 + \ell}{2^n - (q-\ell)}\right) \cdot 2^{-n} , \end{aligned}$$

which implies

$$\frac{\mathrm{p}_0(\tau)}{\mathrm{p}_1(\tau)} \geq 1 - \frac{q\ell + \ell^2}{2^n - (q-\ell)} \geq 1 - \frac{2(q\ell + \ell^2)}{2^n} .$$

□

Lemma 5.33 (Bad event analysis)**.** *For the set $\mathcal{B}$ of bad transcripts (as defined above),*

$$\Pr[T_1 \in \mathcal{B}] \leq \frac{2q}{2^{\gamma^*}} + \frac{q}{2^n} + \frac{q^2}{2^c} + \mathrm{Adv}^{\mathrm{inv},\pi}_{\mathrm{Spg}}(\mathcal{A}) .$$

Proof. The same resampling approach as in the proof of Lemma 5.4 applies here as well. First, observe that a hit occurs if either one of the forward queries returns $s^\star$ or if the attacker makes a backward query on $s^\star$. Hence, the probability of a hit is at most

$$\frac{q}{2^n} + \mathrm{Adv}^{\mathrm{inv},\pi}_{\mathrm{Spg}}(\mathcal{A}) .$$

Consider the following directed graph $G = (V, E)$ based on the query set L:

- the nodes are the capacity parts that appear in L, i.e.,

$$V = \{v^{(\mathrm{c})} \mid (v, *) \in L \vee (*, v) \in L\} ;$$

- two nodes are connected by a labeled edge if a corresponding query has been made, i.e.,

$$E = \{(u^{(\mathrm{c})}, v^{(\mathrm{c})}, l) \mid (u, v) \in L \wedge l = u^{(\mathrm{r})} \oplus v^{(\mathrm{r})}\} .$$

L is called *collision-free* if there is at most one path *with unique labels* from $s_0^{(c)}$ to every other node in G. Let $\mathcal{E}$ be the event that L is *not* collision-free. Note that

$$\Pr[T_1 \in \mathcal{B}] \;\leq\; \Pr[T_1 \in \mathcal{B}|\mathcal{E}] + \Pr[\overline{\mathcal{E}}] \;\leq\; \Pr[T_1 \in \mathcal{B}|\mathcal{E}] + \frac{q^2}{2^c}\,.$$

Towards bounding $\Pr[T_1 \in \mathcal{B}|\mathcal{E}]$, consider a triple $z = (\sigma, L, s_0) \in \mathcal{E}$, which, once more, is shorthand for L being collision-free. Define a *potential chain* to be, for *some* ℓ, any sequence $(u_1, v_1), \ldots, (u_\ell, v_\ell) \in L$ such that $u_1 = s_0 \oplus \mu_1 \| 0^c$ and

$$u_i = v_{i-1} \oplus \mu_i \| 0^c$$

for $i = 2, \ldots, \ell - 1$ and some values $\mu_1, \ldots, \mu_\ell$. Note that $(u_1^{(c)}, v_1^{(c)}), \ldots, (u_\ell^{(c)}, v_\ell^{(c)})$ describe a path from $s_0^{(c)}$ to $v_\ell^{(c)}$ with labels $\mu_1, \ldots, \mu_\ell$. Hence, that for a collision-free L, there are at most q potential chains.

Clearly, conditioned on $Z = z$, $T_1 \in \mathcal{B}$ if and only if for some potential chain, $X_i = \mu_i$ for all $i = 1, \ldots, \ell$. Hence, by the legitimacy of $\mathcal{A}$,

$$\Pr[T_1 \in \mathcal{B}|Z = z] \;\leq\; q \cdot p_z\,,$$

where $p_z := \mathrm{Pred}(\overline{X}|Z = z)$. The remainder of the proof proceeds in the exact same fashion as the proof of Lemma 5.4. □

5.3.1.2 Maintaining Security

The maintaining security of the Sponge PRNG construction Spg is proved along similar lines as that of the previous constructions. This section discusses the differences.

Lemma 5.34 (Maintaining security)**.** *The advantage of any (q, ℓ)-attacker $\mathcal{A}$ against maintaining*

security is bounded by

$$\mathrm{Adv}^{\mathrm{mtn},\pi}_{\mathsf{Spg}}(\mathcal{A}) \;\leq\; \varepsilon^{\mathrm{mtn}}_{\mathsf{Spg}}(q,\ell) \;:=\; \frac{2(q\ell+\ell^2)}{2^n} + \frac{2q}{2^n} + \mathrm{Adv}^{\mathrm{inv},\pi}_{\mathsf{Spg}}(\mathcal{A})\;.$$

In the following, for convenience, let

$$\delta^{\mathrm{mtn}}_{\mathsf{Spg}}(\gamma^*,\ell,q) \;:=\; \frac{2(q\ell+\ell^2)}{2^n} + \frac{2q}{2^n}\;.$$

Proof. Similarly to the preceding maintaining proofs, maintaining security of Spg is shown via an H-coefficient proof. Once more, one considers transcripts

$$\tau = (s^\star, s_0, x_1, \ldots, x_\ell, L)\;,$$

where L refers to π-queries here. In a *bad* transcript, L contains a query of the type

- $(s_0, *)$, which happens (with in the case $b = 1$) with probability at most $|L|/2^n$ since s_0 is completely independent of $\mathcal{A}$'s view, or
- $(*, s^\star)$, which happens (in the case $b = 1$) with probability $|L|/2^n$ via a forward query or with probability $\mathrm{Adv}^{\mathrm{inv},\pi}_{\mathsf{Spg}}(\mathcal{A})$ via a backward query.

As for good transcripts

$$\mathrm{p}_1(\tau) \;=\; 2^{-2n}\cdot p_L \quad\text{and}\quad \mathrm{p}_0(\tau) \;\geq\; \left(1 - \frac{2(q\ell+\ell^2)}{2^n}\right)\cdot 2^{-2n}\cdot p_L\;,$$

where the latter follows via an argument similar to that in the proof of Lemma 5.32. □

5.3.1.3 Next Security

Recall that the next function next of the Sponge construction computes, on an input state s_0,

$$(s, y) \;=\; \mathsf{next}^{\pi}(s_0) \;=\; (\pi(s_0) \oplus 0^r \| s_0^{(\mathrm{c})}, s_0^{(\mathrm{r})}) \,,$$

where s is the new state and y is the output. Next security demands that if s_0 is chosen uniformly at random, then the output of next be indistinguishable from U_{n+r} to an attacker $\mathcal{A}$ making at most to q queries to π. Denote by $\mathrm{Adv}^{\mathrm{next},\pi}_{\mathrm{Spg}}(\mathcal{A})$ the advantage of $\mathcal{A}$.

Lemma 5.35 (Next security)**.** *The advantage of any q-attacker $\mathcal{A}$ against next security is bounded by*

$$\mathrm{Adv}^{\mathrm{next},\pi}_{\mathrm{Spg}}(\mathcal{A}) \;\leq\; \varepsilon^{\mathrm{next}}_{\mathrm{Spg}}(q) \;\coloneqq\; \frac{2q}{2^c} \,.$$

Proof. For a simple H-coefficient proof, consider a transcript

$$\tau = (s_0^{(\mathrm{c})}, s^{\star}, y^{\star}, L),$$

where s_0 is the initial state, $s^{\star}$ is the new state, $y^{\star}$ is the output value, and L are the queries to π. A bad transcript is a transcript with a query of the type

- $(y^{\star} \| s_0^{(\mathrm{c})}, *) \in L$ or
- $(*, s^{\star} \oplus 0^r \| s_0^{(\mathrm{c})}) \in L$.

Since the view of $\mathcal{A}$ in the ideal world, where $\mathcal{A}$'s view is independent of $s_0^{(\mathrm{c})}$, the probability of a bad transcript is at most $2q/2^c$.

For a good transcript, observe that

$$\mathsf{p}_1(\tau) \;=\; 2^{-c} \cdot 2^{-r} \cdot 2^{-n} \cdot p_L \,,$$

where p_L denotes the probability that a uniform random permutation is consistent with the queries in L. Moreover,

$$\mathrm{p}_0(\tau) \;=\; 2^{-c} \cdot 2^{-r} \cdot q_\tau \cdot p_L \;.$$

Note that since τ is a good transcript, $(y^\star, s_0^{(\mathrm{c})})$ is a fresh input to π and $s \oplus 0^r \| s_0^{(\mathrm{c})}$ is still available; hence $q_\tau \geq 2^{-n}$. □

5.3.1.4 Recovering Security

In the following, let $\varepsilon^{\mathrm{ext}}_{\mathrm{Spg}}(\gamma^*, q, \ell)$ and $\varepsilon^{\mathrm{next}}_{\mathrm{Spg}}(q)$ be as in Lemmas 5.31 and 5.35. Once more, extraction and next security together imply recovering security:

Lemma 5.36 (Recovering Security)**.** *For every γ^*-legitimate (q, ℓ)-attacker $\mathcal{A}$,*

$$\mathrm{Adv}^{\mathrm{rec},\pi}_{\mathrm{Spg}}(\mathcal{A}) \;\leq\; \delta^{\mathrm{ext}}_{\mathrm{Spg}}(\gamma^*, \ell, q + r/n + 1) + \frac{q + r/n + 1}{2^n} + 2 \cdot \varepsilon^{\mathrm{next}}_{\mathrm{Spg}}(q + r/n + 1) \;.$$

Proof. As in the proof of Lemma 5.7 consider, for $b \in \{0, 1\}$, the recovering experiment H_b conditioned on the secret bit having the value b. Moreover, again define a hybrid experiment $H_{\frac{1}{2}}$ in which the challenge oracle returns $\mathrm{next}^\pi(U_n)$ to $\mathcal{A}$.

- The distance between H_0 and $H_{\frac{1}{2}}$ is bounded by a similar reduction $\mathcal{A}_{\mathrm{ext}}$ to extraction security as in Lemma 5.7. Hence, it is at most

  $$\varepsilon^{\mathrm{ext}}_{\mathrm{Spg}}(\gamma^*, \ell, q + r/n + 1) \;\leq\; \delta^{\mathrm{ext}}_{\mathrm{Spg}}(\gamma^*, \ell, q + r/n + 1) + \mathrm{Adv}^{\mathrm{inv},\pi}_{\mathrm{Spg}}(\mathcal{A}_{\mathrm{ext}}) \;.$$

  In order to bound the probability of a hit in the ideal world of extraction security, one analyses this event in a hybrid world similar to H_1. By next security and the fact that hit occurs with probability at most $(q + r/n + 1) \cdot 2^{-n}$ in H_1 in that hybrid,

  $$\mathrm{Adv}^{\mathrm{inv},\pi}_{\mathrm{Spg}}(\mathcal{A}_{\mathrm{ext}}) \;\leq\; \frac{q + r/n + 1}{2^n} + \varepsilon^{\mathrm{next}}_{\mathrm{Spg}}(q + r/n + 1) \;.$$

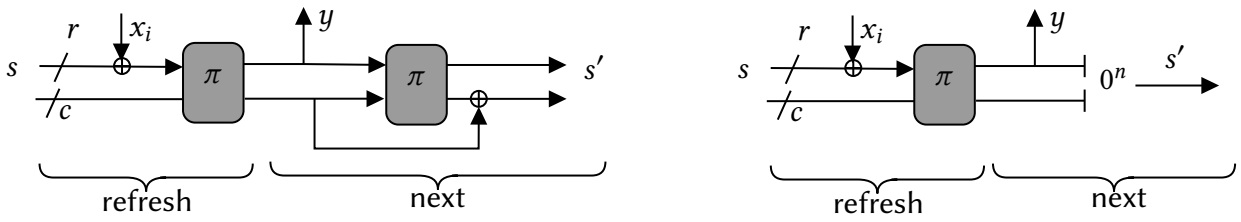

Figure 5.3: Procedures refresh (processing a single-block input x_i) and next of Merkle-Damgård PRNG constructions with compression function F. Left: Computationally secure Construction 10; right: IT candidate Construction 11.

- By an argument similar to that in Lemma 5.7, one uses next security to show that the advantage of $\mathcal{A}$ in distinguishing hybrids $H_{\frac{1}{2}}$ and H_1 is upper-bounded by $\varepsilon^{\text{next}}_{\text{MD}}(q)$.

□

5.3.1.5 Preserving Security

In the following, let $\varepsilon^{\text{mtn}}_{\text{Spg}}(q,\ell)$ and $\varepsilon^{\text{next}}_{\text{Spg}}(q)$ be as in Lemmas 5.31 and 5.35. Once more, maintaining and next security together imply preserving security:

Lemma 5.37 (Preserving Security)**.** *For every (q,ℓ)-attacker $\mathcal{A}$,*

$$\text{Adv}^{\text{pre},\pi}_{\text{Spg}}(\mathcal{A}) \;\leq\; \delta^{\text{mtn}}_{\text{Spg}}(\ell, q + r/n + 1) + \frac{q + r/n + 1}{2^n} + 2 \cdot \varepsilon^{\text{next}}_{\text{Spg}}(q + r/n + 1)\ .$$

The proof proceeds similarly to that of Lemma 5.36—except that reductions to maintaining security are made instead of extraction security—and is omitted.

5.3.2 IT PRNGs from Sponges

In the IT variant of the Sponge construction, refresh remains the same, but next will truncate the input state to r bits, which it outputs, and then zero out the state.

Construction 15 (IT-PRNG from Sponges)**.** The Sponge-based PRNG construction $\mathsf{Spg}_r = (\mathsf{refresh}, \mathsf{next})$ uses a permutation $\pi : \{0,1\}^n \to \{0,1\}^n$ to absorb and produce r-bit inputs and outputs, respectively, and is defined as follows:

- $\mathsf{refresh}^\pi(s, x) = \pi(s \oplus x \| 0^c)$, and
- $\mathsf{next}^\pi(s) = (0^n, s[1..r])$.

An IT-PRNG based on the Sponge paradigm is a modification of the computational variant where the output of next is merely the truncated state to the first r bits along with 0^n as the new state.

Theorem 5.38 (IT-Robustness of Sponge PRNGs)**.** *Construction 15 is a $(\gamma^*, q, t, \ell, \varepsilon_{\text{rob}})$-IT-robust PRNG in the π-model for*

$$\varepsilon_{\text{rob-it}} \leq \frac{t}{2}\sqrt{\frac{2^{r-\gamma^*}}{(1-\rho)} + \frac{\ell \cdot (\ell + \tilde{q})}{2^{c-1}}} + t\rho\,,$$

for $\rho = \frac{\tilde{q}^2}{2^c}$ where $\tilde{q} = q + t\ell$

In this section, we take a look at the proof of robustness for the IT-PRNG based on the Sponge construction.

Lemma 5.39. *For every γ^*-IT-legitimate (q, ℓ)-attacker, in the ideal permutation model,*

$$\mathsf{SD}\left((\tilde{Y}_\ell, \pi)|\mathcal{E}, (U_r, \pi)|\mathcal{E}\right) \leq \frac{1}{2}\sqrt{\frac{2^{r-\gamma^*}}{(1-\rho)} + \frac{\ell \cdot (\ell + q)}{2^{c-1}}} + \rho\,.$$

where $\rho = \frac{q^2}{2^c}$

Proof (of Lemma 5.39). We define a few random variables which we will be using in our proofs.

- π: a randomly chosen permutation, to which the adversary is given access. We use π both for the oracle itself, as well as for the random variable describing the entire function table.

- ℓ: Number of blocks input to the challenge oracle (which is a random variable itself, we overload notation here, using the same letter we use in the bound on ℓ in the lemma statement).
- $\overline{X} = (\overline{X}_1, \ldots, \overline{X}_\ell)$: the blocks input to the challenge oracle.
- $\tilde{Y}_\ell$: output of MD_r. Let us remind ourselves that this is s truncated to r bits (the output of next);
- $Z = (\Sigma, \mathcal{L}, S_0)$: "side information" where Σ is the attacker state before challenge, $\mathcal{L}$ is the attacker query/answers to π before challenge , S_0 is the initial PRNG state provided by $\mathcal{A}$;
- U_r: uniform r-bit string.

The advantage of adversary $\mathcal{A}$ in the recovering game is bounded by $\mathsf{SD}\left(\left(\tilde{Y}_\ell, Z, \pi\right), (U_r, Z, \pi)\right)$. Therefore, it is sufficient to upper-bound that. The reasoning is quite simple. Note that Z contains the state of the attacker Σ just before it makes the challenge query. This means that $\mathcal{A}$ cannot tell apart real from random. For ease of this discussion we define the idea of supernode. State nodes have values in $\{0,1\}^n$. They are clustered to supernodes where a super node with label in $\{0,1\}^c$ contains all state nodes having that same values in the last c bits. Therefore, there are 2^c supernodes with each having 2^r state nodes.

We also define an event $\mathcal{E}$ where the queries in L land do not collide on the last c bits of other elements. This is true even for inverse queries, i.e $\pi^{-1}(v)$ should result in a new supernode. We would like to point out that $\mathcal{E}$ has the same probability of occurring in either experiment, since the experiments are identical up to the point when this event is defined. Therefore, by Proposition 3.4,

$$\mathsf{SD}\left((\tilde{Y}_\ell, Z, \pi), (U_r, Z, \pi)\right) \;\leq\; \mathsf{SD}\left((\tilde{Y}_\ell, Z, \pi)|\mathcal{E}, (U_r, Z, \pi)|\mathcal{E}\right) + \frac{q^2}{2^c}\;;$$

For convenience we let $\rho := \frac{q^2}{2^c}$ for the remainder of the proof. In order to bound the statistical distance conditioned on $\mathcal{E}$, we can rewrite the same as as

$$\mathsf{SD}\left((\tilde{Y}_\ell, Z, \pi)|\mathcal{E}, (U_r, Z, \pi)|\mathcal{E}\right) \;=\; \sum_{z\in\mathcal{E}} \Pr[Z = z|\mathcal{E}] \cdot \mathsf{SD}\left((\tilde{Y}_\ell, \pi)|z, (U_r, \pi)|z\right) , \tag{5.10}$$

where $z \in \mathcal{E}$ is to denote that the sum is taken over all side informations $Z = z$, satisfying $\mathcal{E}$.[12] Define $p_z := \mathsf{Pred}(\overline{X}|Z = z)$, and observe that

$$\mathbf{E}_z[p_z] \;=\; \mathsf{Pred}(\overline{X}|Z) \;\leq\; 2^{-\gamma^*} ,$$

where the latter inequality follows from the assumption $\mathrm{H}_\infty(\overline{X}|Z) \geq \gamma^*$. Moreover,

$$\mathrm{H}_\infty(\overline{X}|Z\mathcal{E}) \;\geq\; \gamma^* - \log(1-\rho)^{-1} ,$$

which is due to

$$\begin{aligned}
\mathsf{Pred}(\overline{X}|Z) \;&\geq\; \sum_{z\in\mathcal{E}} \Pr[Z = z] \cdot p_z \\
&=\; \Pr[\mathcal{E}] \cdot \sum_{z\in\mathcal{E}} \frac{\Pr[Z = z]}{\Pr[\mathcal{E}]} \cdot p_z \\
&=\; \Pr[\mathcal{E}] \cdot \mathsf{Pred}(\overline{X}|Z\mathcal{E}) .
\end{aligned}$$

From Lemma 5.40 we prove below, we will get,

$$\mathsf{SD}\left((\tilde{Y}_\ell, \pi)|z, (U_r, \pi)|z\right) \;\leq\; \frac{1}{2}\sqrt{2^r p_z + \frac{\ell \cdot (\ell + q)}{2^{c-1}}} \quad .$$

[12] Therefore $\mathcal{E}$ can be omitted in the conditioning of the statistical distance.

Using Jensen's inequality, (5.10) becomes, for

$$\begin{aligned}\mathsf{SD}\left((\tilde{Y}_\ell,\pi)|\mathcal{E},(U_r,\pi)|\mathcal{E}\right) &\leq \frac{1}{2}\sqrt{2^r\mathsf{Pred}(\overline{X}|\mathcal{LE})+\frac{\ell\cdot(\ell+q)}{2^{c-1}}}\\ &\leq \frac{1}{2}\sqrt{\frac{2^{r-\gamma^*}}{(1-\rho)}+\frac{\ell\cdot(\ell+q)}{2^{c-1}}}\,.\end{aligned}$$

□

Lemma 5.40. *For $z\in\mathcal{E}$ and the random variables as defined earlier,*

$$\mathsf{SD}\left((\tilde{Y}_\ell,\pi)|z,(U_r,\pi)|z\right) \leq \frac{1}{2}\sqrt{2^r p_z+\frac{\ell\cdot(\ell+q)}{2^{c-1}}}\,.$$

Proof. We fix $z=(\sigma,L,s_0)$. We have that z, π is distributed uniformly over the set of all ideal permutations that agree with L. Thus, by Proposition 3.3,

$$\mathsf{SD}\left((\tilde{Y}_\ell,\pi)|z,(U_r,\pi)|z\right) \leq \frac{1}{2}\sqrt{2^r\cdot\mathsf{Coll}(\tilde{Y}_\ell|\pi z)-1}\,. \tag{5.11}$$

We consider the following experiment to bound the collision probability:

- choose π uniformly consistent with L
- sample inputs $\overline{X}=(\overline{X}_1,\ldots,\overline{X}_\ell)$ and $\overline{X}'=(\overline{X}'_1,\ldots,\overline{X}'_{\ell'})$ independently but conditioned on $Z=z$.
- compute $\tilde{Y}_\ell$ and $\tilde{Y}'_\ell$ as the Sponge evaluation of $\overline{X}$ and $\overline{X}'$

Now, we have that:

$$\Pr[\tilde{Y}_\ell=\tilde{Y}'_\ell] \leq \Pr[\overline{X}=\overline{X}'] + \Pr[\tilde{Y}_\ell=\tilde{Y}'_\ell|\overline{X}\neq\overline{X}']\,. \tag{5.12}$$

By definition, we have that the former term is p_z. We take a closer look at the latter term. We fix arbitrary inputs $\overline{x}$ and $\overline{x}'$. Let the length of them be ℓ and ℓ' respectively. We also assume, wlog,

that the evaluation of $\overline{x}'$ is not completely covered by L; due to the collision-freeness of L. Let $\overline{x}_{k+1}$ be the first block of $\overline{x}$ not covered by $\mathcal{L}$ and similarly $\overline{x}'_{k+1}$ for $\overline{x}'$. We let $k = \ell$ if all blocks are covered. We then apply Lemma 5.41 to conclude the proof. □

Lemma 5.41. *Let π be sampled randomly consistent with L, and $\overline{x} \neq \overline{x}'$. Then, let $\tilde{Y}_\ell$ and $\tilde{Y}'_\ell$ be values obtained after truncating at the end of the Sponge evaluations on inputs $\overline{x}$ and $\overline{x}'$ respectively. Then,*

$$\Pr[\tilde{Y}_\ell = \tilde{Y}'_\ell] \leq \frac{\tilde{\ell}(q+\tilde{\ell})}{2^{n-1}} + \frac{1}{2^r} .$$

Proof. We denote by $\mathsf{Coll}_L(\overline{x}, \overline{x}')$ the event that the sponge evaluation on inputs $\overline{x}, \overline{x}'$ collide at the final state node, for a permutation π sampled consistently with L. We rewrite $\Pr[\tilde{Y}_\ell = \tilde{Y}'_\ell]$ as:

$$\Pr[\tilde{Y}_\ell = \tilde{Y}'_\ell] \leq \Pr[\mathsf{Coll}_L(\overline{x}, \overline{x}')] + \Pr[\tilde{Y}_\ell = \tilde{Y}'_\ell | \overline{\mathsf{Coll}_L(\overline{x}, \overline{x}')}]$$

We take a closer look at the latter term. Note that the evaluation of the inputs is fixed and it has no collision at the end nodes. We proceed to assign random, yet distinct values to the final state nodes. These are chosen from $\{0,1\}^n$. Note that it is sufficient to look at the two output vertices *locally* without looking at the global state. There are $2^n(2^n-1)$ pairs of values for these output vertices such that these values are distinct. However, of these, $2^r 2^{n-r}(2^{n-r}-1)$ have values which are equal in the first r bits and yet are distinct n-bit strings. Therefore,

$$\begin{aligned}
\Pr[\tilde{Y}_\ell = \tilde{Y}'_\ell | \overline{\mathsf{Coll}_L(\overline{x}, \overline{x}')}] &\leq \frac{2^r 2^{n-r}(2^{n-r}-1)}{2^n(2^n-1)} \\
&= \frac{2^{n-r}-1}{2^n-1} \\
&\leq \frac{2^{n-r}}{2^n} = \frac{1}{2^r} .
\end{aligned}$$

Putting it together, we have:

$$\Pr[\tilde{Y}_\ell = \tilde{Y}'_\ell] \leq \Pr[\mathsf{Coll}_L(\overline{x}, \overline{x}')] + \frac{1}{2^r} \ . \tag{5.13}$$

Now we bound the value of $\Pr[\mathsf{Coll}_L(\overline{x}, \overline{x}')]$. Consider the intermediate evaluations starting from $k+1$. These are the inputs not covered by the list L. Define them as: $U_{k+1}, V_{k+1}, \ldots, U_{\tilde{\ell}}, V_{\tilde{\ell}}$ such that:

$$U_{k+1} = s_k \oplus \overline{x}_{k+1} || 0^c$$

and for $i = k+2, \ldots, \tilde{\ell}$,

$$U_i = V_{i-1} \oplus x_i || 0^c \ .$$

In addition, for $i = k+1, \ldots, \tilde{\ell}$

$$V_i = \pi(U_i) \ .$$

For $i = k+1, \ldots, \tilde{\ell}$, denote by $\mathsf{FRESH_i}$ be the event U_i is fresh, i.e, $\pi(U_i)$ has not yet been defined. It is clear that $\mathsf{FRESH_{k+1}}$ is true. Let

$$\mathsf{FRESH} = \bigcap_{i=k+1}^{\tilde{\ell}-1} \mathsf{FRESH_i}$$

Then,

$$\Pr[\mathsf{Coll}_L(\overline{x}, \overline{x}')] \leq \Pr[\mathsf{Coll}_L(\overline{x}, \overline{x}') | \mathsf{FRESH}] + \Pr[\overline{\mathsf{FRESH}}]$$

Clearly the former term is,

$$\Pr[\mathsf{Coll}_L(\overline{x}, \overline{x}') | \mathsf{FRESH}] \leq \frac{1}{2^n - \tilde{\ell} - q}$$

Given that $\mathsf{FRESH_{i-1}}$ is true, this implies that $V_{i-1} = \pi(U_{i-1})$ was freshly chosen. Therefore, for $\mathsf{FRESH_i}$ to be false, V_{i-1} must have been chosen such that $U_i = V_{i-1} \oplus x_i$ is not fresh. Remember

that, V_{i-1} would have been uniformly sampled from at least $2^n - q - \tilde{\ell}$ values. Therefore,

$$\Pr[\overline{\mathsf{FRESH}_i} | \bigcap_{j=k+1}^{i-1} \mathsf{FRESH}_j] \leq \frac{(q+i)}{2^n - \tilde{\ell} - q} .$$

In other words, $\Pr[\overline{\mathsf{FRESH}}] \leq \frac{(\tilde{\ell}-1)(q+\tilde{\ell}-1)}{2^{n-1}}$. □

5.3.3 Parameter Choices

In terms of concrete parameters, observe the following for the PRNG constructions from Sponges above: above:

- **Computational PRNG:** SHA-3 like parameters have $n = 1600$ and $c = 1024$. We let $t = 1, q = 2^{80}$ and let $\ell = \gamma^*$. This assumes that we get at least one bit of entropy from each block. We would need $\gamma^* \approx 163$ to get 80 bits of security.
- **IT PRNG:** We let $n = 1600$ and $c = 1024$. In addition, we let $t = 1$ and $q = 2^{80}$. We also let $\ell = \gamma^*$. Therefore, we incur an entropy loss of 160 bits to get 80 bits of security.

6 On Seedless PRNGs and Premature Next

This chapter is based on joint work with Sandro Coretti, Yevgeniy Dodis, Noah Stephens-Davidowitz, and Stefano Tessaro that appeared in ITC 2022 [CDK+22]. Passages are taken verbatim from this paper. The chapter considers the security of seedless PRNGs against *premature next attacks* [KSWH98]. The idea behind such an attack is that next—the algorithm extracting pseudorandom bits from the PRNG state—is called before the state has accumulated sufficient entropy. In Section 6.1, we look at the problem in greater detail and discuss the current state of the art, prior to this dissertation. In Section 6.2, we see an impossibility result for the existence of seedless PRNGs which are "premature-next" robust. In Section 6.3, we define seedless schedulers and motivate a need for relaxing the security definition. In Section 6.4 we look at one such relaxation and in Section 6.5 we look at the other relaxation. In both these cases, we show a lower-bound for security and present constructions that match this lower bound.

6.1 The Premature Next Problem

The resulting output will therefore not be fully random, and an adversary can potentially use the output of many such calls to recover the state. We adopt the notion of robustness against premature-next attacks as defined by Dodis *et al.* [DSSW17]. Their work generalized and ana-

lyzed a key technique to mitigate such attacks that originated in the designs of the Yarrow [KSF99] and Fortuna [FS03] PRNGs. Roughly, the key idea is that the entropic inputs to the PRNG are carefully distributed to several "smaller" PRNGs, which we refer to as *pools*, and, with different frequencies, these pools are used to randomize a *register* from which random bits are extracted. (We formalize this approach in detail below.) While both Yarrow and Fortuna use deterministic *scheduling* strategies to assign entropic inputs to a pool and to decide when each pool contributes to the register, the provable robustness against premature-next attacks is achieved in [DSSW17] by relying on a *random seed* (independent from the inputs) to ensure that the entropy received from the adversary is roughly evenly distributed among the pools.

It is not hard to see that the fixed pool assignment schedule adopted by Yarrow/Fortuna cannot be robust against premature next attacks without extreme restrictions on the adversaries (e.g., the constant rate restriction). However, other seedless strategies are possible (e.g., one could assign entropic inputs to pools chosen depending on the inputs themselves, or some previous inputs; or one might try to divide each input up into smaller pieces in some way; or one might not use pools at all), and the larger question remains on the feasibility of a *seedless* PRNG which is robust, even with premature next calls. One of course should exercise some care, a fully secure deterministic PRNG cannot exist (regardless of premature-next attacks) for the same reasons deterministic extraction is impossible. So, we must make some restrictions on the input distributions provided by the adversary. For this reason, in the following section, we will focus on the case of *independent* inputs, for which deterministic extraction is–in principle–possible.

6.2 Impossibility of "Premature Next" Robust Seedless PRNGs

Even in this setting, the main result of this section is an impossibility result. (So, the fact that we restrict our attention to independent inputs simply makes our result stronger.) Specifically,

we show that it is impossible to have such a seedless PRNG which is robust against premature next attacks, even in a setting where the entropic inputs are independent. Before we present our result, which is stated below as Theorem 6.5, we introduce some more syntax and definitions.

6.2.1 Pseudorandom Number Generators with Input

In this section, we will briefly recall the syntax of this primitive, as discussed in Chapter 4.

Syntax. A PRNG is a stateful cryptographic primitive that accumulates entropy by absorbing inputs which it then uses to produce pseudorandom bits when the entropy of its state is high. A PRNG consists of two algorithms as defined below:

Definition 6.1 (Syntax of PRNGs)**.** A *pseudorandom number generator with input (PRNG)* is a pair of algorithms $\mathsf{PRNG} = (\mathsf{refresh}, \mathsf{next})$ sharing a n-bit state s, where

- $\mathsf{refresh}$ takes a state s and an input $x \in \{0,1\}^m$ and produces a new state $s' = \mathsf{refresh}(s, x)$, and
- next takes a state s and produces a new state and an output $y \in \{0,1\}^r$, i.e, $(s', y) = \mathsf{next}(s)$.

A PRNG processing m-bit inputs and producing r-bit output is called a (m, r)*-PRNG*.

For our impossibility result, we will focus on $(1,1)$-PRNG. This is without loss of generality, as we could always buffer m such entropic inputs before applying a "bigger" refresh call on m such bits, and impossibility for 1 output bit implies that for $r \geqslant 1$ output bits.

Security. In Chapter 4, we looked at the robustness security game, *without* support for Premature Next (ROB). For purposes of this paper, we will focus on robustness security *with* Premature Next (NROB), as defined in Figure 6.1. While we adapt the original definition from [DSSW17] to the seedless setting, we note that we present a highly simplified security game that is enough to provide for our impossibility result. (We also leave out some functionality that is not necessary

for the attacker in our impossibility result, which again simply makes our impossibility result stronger.)

Most significantly, we assume that all of the samples provided by the attacker are *independent* from each other (which makes out impossibility much result stronger). Formally, attacker outputs a distribution X_i for the next entropic sample, and the security game independently samples a concrete value $x_i \leftarrow\!\!\$\, X_i$ from this distribution, without giving any side information back to the attacker. This allows for much simpler accounting for entropy, — by simply adding individual entropy of samples X_i produced by the attacker, — without worrying about (quite subtle) conditional entropy of such samples.

In more detail, NROB game allows adversary to access to the following oracles:

- **get-next** allows the attacker to get pseudorandom outputs by calling the next procedure on the current state and returning the output y.
- **next-ror** creates a challenge, i.e., if $b = 1$, it outputs a uniform random value $y_1 \in \{0,1\}$ instead of the PRNG output y_0. Here, the PRNG output is second part of the output of next procedure.
- **get-state** models state compromises by revealing the value of the state.

Definition 6.2 (Definition of an Attacker)**.** An attacker $\mathcal{A}$ is called a (q, τ)-attacker if it provides at most q input distributions for refresh and runs in time at most τ.

For security, the game keeps track of the entropy counter c which counts the entropy the attacker injected into the system since the latest compromise. When c reaches a critical value γ^*, we would like our PRNG to recover. However, instead of demanding immediate recovery (like in the simpler robustness game ROB discussed in Chapter 4), we allow a factor of β gap. Concretely, if entropy γ^* took T^* steps to accumulate, we demand recovery by time $T \leqslant \beta T^*$.

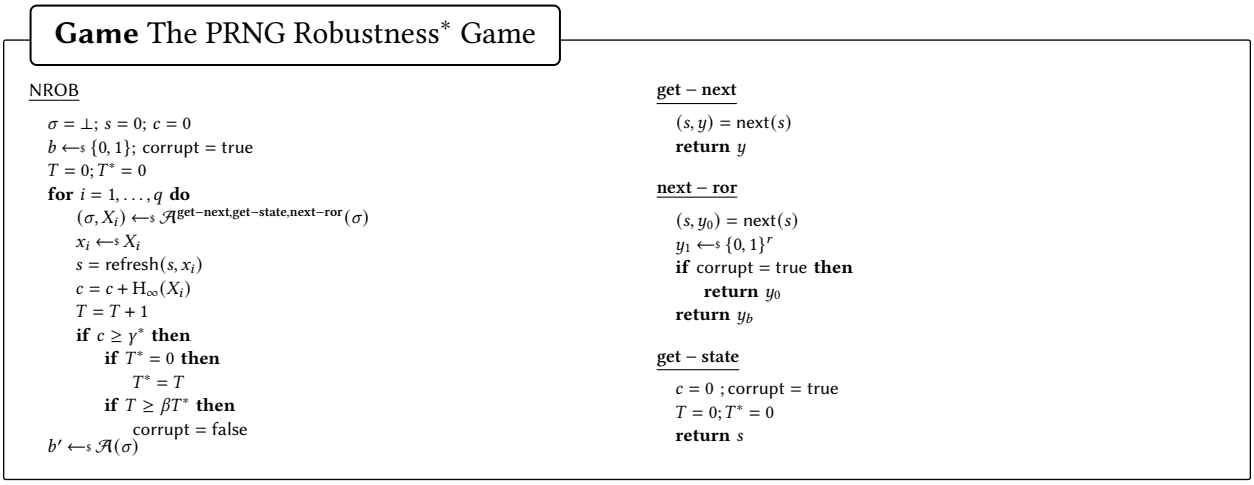

Figure 6.1: The Robustness Game with Premature Next Calls NROB(γ^*, β, q).

Definition 6.3. The advantage of a (q, τ)-attacker $\mathcal{A}$ in the NROB(γ^*, β, q) game is denoted by $\mathrm{Adv}^{\mathrm{NROB}}_{\mathrm{PRNG}}(\mathcal{A})$. Further, we say that PRNG is $(\gamma^*, \beta, q, \epsilon, \tau)$-secure if for any (q, τ)-attacker $\mathcal{A}$,

$$\mathrm{Adv}^{\mathrm{NROB}}_{\mathrm{PRNG}}(\mathcal{A}) \leq \epsilon$$

6.2.2 Impossibility Result

The idea of our attack is that the adversary provides bit inputs such that every n inputs has one bit of entropy. Further, the premature next call will reveal information about this bit. We will prove the result through a series of lemmas. As mentioned before, we will assume that the inputs and the outputs are merely bits.

In the remainder of this section, we will work with a function $f_{\mathrm{PRNG}} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, for PRNG $=$ (refresh, next). This function $f_{\mathrm{PRNG}}(s, x)$ represents the application of n iterated refresh calls, starting from an initial state s with input $x_1, \ldots, x_n \in \{0,1\}$, before finally applying next to produce an output bit y, or more formally:

$$
\begin{aligned}
&f_{\mathsf{PRNG}}(s, x_1 || \ldots || x_n):\\
&\qquad \textbf{for } i = 1 \textbf{ to } n\\
&\qquad\qquad s = \mathsf{refresh}(s, x_i)\\
&\qquad (s, y) = \mathsf{next}(s)\\
&\qquad \textbf{return } y
\end{aligned}
$$

This is equivalent to applying one "big-refresh" before one next, as indicated before. Further, we write x_{-i} for $x_1 || \ldots || x_{i-1} || x_{i+1} || \ldots || x_n$, i.e., the binary string x, except for the i-th bit. Then, we can define $x_{-i,\chi}$ to be the string where the i-th bit is set to χ, i.e.

$$x_{-i,\chi} := x_1, \ldots, x_{i-1}, \chi, x_{i+1}, \ldots, x_n$$

For any function g and any i, we abuse notation and write $g(x_{-i,\chi})$ as a shorthand for $g(x_1 || \ldots || x_n)$ where i-th bit is χ. We will also use X to denote the random variable corresponding to $x_1 || \ldots || x_n$ and use X_{-i} to denote the random variable corresponding to x_{-i}.

Lemma 6.4. *There exists a randomized $O(n^2)$ algorithm $\textsc{Find}^g$ with oracle access to any function $g : \{0,1\}^n \to \{0,1\}$, such that with probability at least $1 - 2^{-n}$ (over the coins $\textsc{Find}^g$), $\textsc{Find}^g$ outputs (i, z) which satisfies precisely one of the following two (disjoint) properties:*

- *$i = 0$, $z \in \{0,1\}$, and $\Pr[g(U_n) = z] \geq 0.6$.*
- *$1 \leq i \leq n$, $z \in \{0,1\}^{n-1}$ and $g(x_{-i,0}) \neq g(x_{-i,1})$ where $x_{-i} = z$.*

(In other words, $\textsc{Find}^g$ either discovers that $g(U_n)$ is biased, or it identifies two n-bit strings that differ in a single bit such that g returns different values on these two strings.)

Proof. The algorithm $\textsc{Find}^g$ is defined in Figure 6.2. The $\textsc{Find}^g$'s output satisfies case 2 unless , after n^2 tries, the algorithm fails to find a value in the first loop. Further, in the second loop, the algorithm merely outputs the majority element.

Analysis of First **for** Loop. Let us look at trying to determine i, z such that it satisfies the second property. To this end, we will rely on results from graph theory. Specifically, we will use the edge isoperimetric inequality for a Hypercube graph [HLW06, §4], which we recall (in our context) below.

For our setting, we have a Hypercube graph $Q_n = (V, E)$ where each vertex corresponds to a binary vector of length n, i.e., $|V| = 2^n$. Further E is the set of all edges that connects $(\boldsymbol{u}, \boldsymbol{v})$ if the Hamming distance between $\boldsymbol{u}$ and $\boldsymbol{v}$ is exactly 1. This gives us that: $|E| = n \cdot 2^{n-1}$. Now, we are interested in edges between a vertex $\boldsymbol{u}$ and $\boldsymbol{v}$ if $g(\boldsymbol{u}) \neq g(\boldsymbol{v})$. Now, for any set S of size $k \leq 2^{n-1}$, the number of "cut" edges C from the set to its compliment is bounded by the isoperimetric inequality [HLW06, §4.2.1] as follows:

$$C \geq k \cdot (n - \log_2 k) \geqslant k$$

However, now we need to determine how many $\boldsymbol{u} \in V$ exists such that $g(\boldsymbol{u}) = 0$ (or 1). If, $0.4 \leq \mathbb{E}[g(U_n)] \leq 0.6$, then we know that there exists $0.4 \cdot 2^n$ vectors $\boldsymbol{u}$ with $g(\boldsymbol{u}) = 0$ and a similar number for $g(\boldsymbol{u}) = 1$.

Therefore, the probability of choosing the desired edge is at least:

$$\frac{k}{n \cdot 2^{n-1}} \geq \frac{0.4 \cdot 2^n}{n \cdot 2^{n-1}} = \frac{0.8}{n}$$

In other words, the probability that a randomly chosen edge is the desired edge occurs with probability $p \geq 0.8/n$. Therefore, one can simply pick an edge $e \in E$, uniformly at random, and then test to see if it is the desired edge. Now, if one were to do n^2 such tests, we get:

$$\Pr[g(x_{-i,0}) \neq g(x_{-i,1})] > 1 - 2^{-n}$$

Algorithm $\textsc{Find}^g$

for $i = 1$ **to** n^2:
  Pick an edge $(\boldsymbol{u}, \boldsymbol{v}) \in E$, uniformly at random.
  Use oracle access to g to compute $g(\boldsymbol{u})$ and $g(\boldsymbol{v})$.
  if $g(\boldsymbol{u}) \neq g(\boldsymbol{v})$ **then**
    Find i such that $u_i \neq v_i$.
    By definition, there exists a unique i that satisfies this condition.
    return (i, u_{-i})
    break
for $i = 1$ **to** $120 \cdot n$:
  $count = 0$
  Sample $\boldsymbol{x} \leftarrow \{0,1\}^n$
  Compute $count = count + g(\boldsymbol{x})$
if $count > n/2$ **then** $z = 1$
else $z = 0$
return $(0, z)$

Figure 6.2: Description of $\textsc{Find}^g$.

This math follows from the fact that the probability of failure of algorithm is:

$$\left(1 - \frac{0.8}{n}\right)^{n^2} \leq e^{-0.8n} < 2^{-n}$$

Note that this result only follows if $0.4 \leq \mathbb{E}[g(U_n)] \leq 0.6$.

Analysis of Second **for** Loop. However, if $\mathbb{E}[g(U_n)] < 0.4$ or $\mathbb{E}[g(U_n)] > 0.6$, then we know that the distribution, is biased either in favor of 0 or 1. If it is biased in favor of 1 (i.e., $\mathbb{E}[g(U_n)] > 0.6$), then we know that $> 0.6 \cdot 2^n$ inputs $\boldsymbol{x}$ will evaluated to 1 or $< 0.4 \cdot 2^n$. In other words, the probability of success $p > 0.6$. Therefore, one can apply Chernoff bounds, to get that $\Pr[g(U_n) = z] \geq 0.6$ with probability $1 - 2^{-n}$.

The correctness of $\textsc{Find}^g$ follows from our earlier discussion. It is easy to see that $\textsc{Find}^g$ runs in time $O(n^2)$ as the lines inside the first for loop take constant time if one were to sample the edge by picking i and x_{-i}. □

Theorem 6.5. *There is no $(\gamma^*, \beta, q, 0.1, \tau)$-secure PRNG for $\gamma^* \beta < \sqrt{q}$ and $\tau \geq \Omega((t_{\mathsf{next}} + t_{\mathsf{refresh}}) \cdot n^3)$ where $n = \gamma^* \cdot \beta$ and t_{next} and t_{refresh} are the time required to compute* next *and* refresh *respectively.*

Proof. We will use the $\textsc{Find}^g$ algorithm defined in Lemma 6.4 to create an adversary $\mathcal{A}$ that wins the $\mathsf{NROB}(\gamma^*, \beta)$ security game. The pseudocode for the adversary is provided in Figure 6.3. $\mathcal{A}$ will set $g(\boldsymbol{x}) = f(s, x_1 || \ldots || x_n)$ for the current state s and $n = \gamma^* \cdot \beta$. $\mathcal{A}$ is aware of the very first state s. The attacker then runs $\textsc{Find}^g$ on this function g.

$i = 0$. If, $\textsc{Find}^g$ returns (i, z) such that $i = 0$, then $\mathcal{A}$ simply calls **get − state**, and then provides uniform bit $n = \gamma^*\beta$ times. Note that the invocation of **get − state** is critical to reset the counters of T and T^* to 0. This will leave us with $T^* = \gamma^*\beta$ and the attacker is required to break the scheme within another β steps. After the n input distributions, $\mathcal{A}$ then invokes **next − ror** to receive its challenge. If its challenge is equal to z, then we know that $b = 0$, indicating it is the real distribution and not the random distribution.

$i \neq 0$. However, if $i \neq 0$, then the $\mathcal{A}$ writes down z in its state, and then provides the entropy in x_i. It then makes a "premature" call to **get-next** , which reveals the actual input bit x_i, helping $\mathcal{A}$ recover the state. This process is repeated γ^* times to provide γ^* bits of entropy. We keep doing this for $\gamma^{*2}\beta^2$ steps, and then, request **next-ror** . However, with knowledge of the state, due to premature next, $\mathcal{A}$ knows the challenge and therefore wins with a non-negligible advantage.

In other words, we have an attacker which can break this scheme, with non-negligible probability, if $q > \gamma^{*2}\beta^2$. □

Note, that when $q < \gamma^*\beta$, every PRNG is vacuously secure as there is no need for recovery: at least γ^* steps are needed to inject the required γ^* bits of entropy, and the attacker simply runs out of refresh calls to trigger the security requirement. This, of course, assumes ideal entropy accumulation.

6.2.3 Towards Positive Results

The impossibility is, of course, artificial, but it raises questions about how to overcome it, even assuming ideal entropy accumulation and extraction. In Section 6.3 we abstract the no-

Algorithm $\mathcal{A}$

Set $s = 0$
$\sigma = \perp$
$t = 0$
while $t \leq \gamma^{*2}\beta^2$
  Set $g(\boldsymbol{x}) = f(s, \boldsymbol{x})$
  $(i, z) \leftarrow \textsc{Find}^g$
  if $i = 0$ **then**
    Invoke **get-state** to get the current state s^*. // This resets $T = 0$.
    for $j = 1$ **to** n:
      Output $X_{t+j} = U_1$ // $\mathrm{H}_\infty(X_{t+j}) = 1$.
    Invoke **next-ror** for challenge δ
    if $\delta = z$ **then return** 0
    else return 1
  else
    Set $X_{t+i} = U_1$ // $\mathrm{H}_\infty(X_{t+i}) = 1$.
    Use z to set X_{t+k} for $k \neq i$. // $\mathrm{H}_\infty(X_{t+k}) = 0$ for $k \neq i$.
    Invoke **get-next** to get output y.
    Let $a_{-i} = z$
    if $g(a_{-i,0}) = y$ **then** $x_{t+i} = 0$
    else $x_{t+i} = 1$
    for $i = 1$ **to** $\alpha\beta$
      $s = \mathsf{refresh}(s, x_{t+i})$
    $(s, y) = \mathsf{next}(s)$
    $t = t + \alpha\beta$
Invoke **next-ror** for challenge δ
if $\mathsf{next}(s) = (\cdot, \delta)$ **then return** 0
else return 1

Figure 6.3: Pseudocode for $\mathcal{A}$ for Theorem 6.5.

tion of the scheduler which models security against premature next attacks using multiple pools which assume to accumulate entropy optimally (which abstracts away entropy accumulation and extraction).

In this setting, we will first analyze a single-pool scheduler scheme for the special "root pool" in Section 6.4. This scheme uses a single pool with exponentially decaying time intervals to drain this pool, but the rate of such recovery will depend on the entropy rate *counted from the boot time* (as opposed to the latest compromise in the general notion). The latter point is why we don't want to use this one-pool scheme for the general-purpose PRNG, where we would like to recover from compromise *no matter when it happens.*

For such scenarios, we revisit the round-robin Fortuna scheduler, where [DSSW17] observe

that this scheme provably overcomes our impossibility result, by assuming all entropy comes at a fixed (but unknown) rate. Instead, in Section 6.5 we significantly generalize this positive result. The idea is to redefine the notion of entropy we use in a way that makes it more restrictive than traditional (min-) entropy, but not as restrictive as assuming fixed constant rate.[1] Intuitively, our notion of entropy will not allow attacks where the entropy varies too widely within a given round-robin (but can change from one round-robin to another) — in a sense that the attacker will get almost no credit for high-entropy samples when there is at least one low entropy sample within a given round-robin.

6.3 Seedless Scheduler

For the remainder of this paper, we will assume ideal accumulation and extraction. Further, rather than working with entropy, we will employ the notion of a sequence of weights $\boldsymbol{w} = (w_1, \ldots, w_q)$ where the weights have been normalized so that $w_i \in [0, 1]$ and a pool is "full" when it has accumulated weight 1. (Specifically, to move between the weight w_i and the entropy γ, one should multiply by the entropy γ^*_{rob} required for a single pool to recover.) See [DSSW17].

6.3.1 Syntax of a Scheduler

We define the syntax of the scheduler below. Note that this scheduler is deterministic and oblivious, i.e., it does not depend on the actual input or its entropy.

Definition 6.6 (Syntax of Scheduler)**.** A (k, q)-scheduler is a deterministic algorithm SC that produces q pairs: $\{(\mathsf{in}_i, \mathsf{out}_i)\}_{i=1}^{q}$ where $\mathsf{in}_i \in [k], \mathsf{out}_i \in [k] \cup \{\perp\}$ for $i = 1, \ldots, q$.

Note that, when the number of "pools" k is not critical to be specified explicitly, a deterministic (k, q)-scheduling scheduler can be thought of as a sequence of values $\{\mathsf{empty}\}_{i=1}^{q}$ corresponding

[1] We also note that the results about fast entropy accumulation in the register [DGSX21a] might justify why our new (more restrictive) notion of entropy might be reasonable to expect in practice.

Construction: Premature-Next Robust PRNG

refresh$^*(x, \bar{s})$

Parse $\bar{s}$ as $(s_0, \ldots, s_{k-1}, \rho)$
in, out $\leftarrow$ SC()
$s_{\text{in}} \leftarrow \text{refresh}(s_{\text{in}}, x)$
$(s_{\text{out}}, R) \leftarrow \text{next}(s_{\text{out}})$
$\rho \leftarrow \rho \oplus R$
return $\bar{s} = (s_0, \ldots, s_{k-1}, \rho)$

next$^*(\bar{s})$

Parse $\bar{s}$ as $(s_0, \ldots, s_{k-1}, \rho)$
$(Y, \rho) \leftarrow G(\rho)$
return $\left(\bar{s} = (s_0, \ldots, s_{k-1}, \rho), Y\right)$

Figure 6.4: Construction of $G = (\text{refresh}^*, \text{next}^*)$.

to the time at which each input i with weight w_i is emptied. More formally, we can define:

$$\text{empty}_i := \min \{j : j > i \wedge \text{out}_j = \text{in}_i\}$$

6.3.2 Seedless PRNG, with Premature Next

Before we venture into the security of such a scheduler, it would be prudent to take a step back and look at an informal composition of a seedless scheduler with PRNGs that are not resilient to premature next in order to achieve security with premature next. Indeed, it is also equally important to frame our composition results, in the face of the impossibility result from Section 6.2.2 (and also the unrestricted scheduler impossibility later in this section). This is precisely the reason why we do not state a formal composition theorem, as it is vacuous for the most general case. However, the composition is still robust for restricted notions of scheduler security to yield relaxed forms of PRNG security with premature next.

The composition relies on seedless PRNGs which are not secure with premature next. These are typically parametrized by just γ^*, which is the minimum entropy needed for the PRNG to begin producing pseudorandom outputs (as discussed in Chapter 4). In essence, these have $\alpha = \gamma^*$ and $\beta = 1$ with a reset of all counters when an adversary invokes **get** − **next** with corrupt = true. The instantiation of this PRNG can be any of the constructions from Chapter 4 or from the work of Dodis *et al.* [DGSX21b]. Such a PRNG, secure without premature next and parametrized by

γ^* is combined with a scheduler. The goal of a scheduler would be to ensure that the input, as it arrives, is allocated a particular pool such that:

- With "enough entropy", a pool is filled, i.e., accumulates γ^* amount of entropy.
- This pool will be emptied within "sufficient time", to recover from compromise.

We will formalize these notions of "enough entropy" and "sufficient time" in the next section.

Formally, we define a seedless PRNG, with construction as follows:

- Let SC be a scheduler with k pools.
- Let $G_i = (\mathsf{refresh}_i, \mathsf{next}_i)$ be *seedless* PRNGs with input, for $i = 0, \ldots, k-1$. For simplicity, we will assume that each G_i is (m, r)-PRNG. These are PRNGs which are not secure with premature next calls, as seen in Chapter 4.
- Let $\boldsymbol{G} : \{0,1\}^m \longrightarrow \{0,1\}^{2m}$ be a pseudorandom generator (without input).

Then, we construct a PRNG with input $G(\mathrm{SC}, \{G_i\}_{i=0}^{k-1}, \boldsymbol{G}) = (\mathsf{refresh}^*, \mathsf{next}^*)$ as shown in Figure 6.4, where the scheduler mandates which pool G_{in} to use (via refresh) to accumulate entropy from a new sample, and which pool G_{out} (if any) to "empty" (via next) into the main register ρ for $\boldsymbol{G}$.

6.3.3 Security of a Scheduler

We will define different notions of security for a scheduler. As with PRNGs, (k, q)-scheduler security model is parameterized by two parameters α, β. Informally, it states that if the adversary chooses to provide α units of fresh entropy (i.e., a sequence of w_i values that sum up to α) within a time $t \leq q/\beta$, then we guarantee recovery within time $\beta \cdot t \leq q$. Formally,

Definition 6.7 (General Security of Scheduler)**.** A (k, q)-scheduler is (α, β)-general-secure if if $\forall t_0, t$ such that $t_1 = t_0 + \beta \cdot t \leq q$, and $\forall$ weights $w_1, \ldots, w_q \in [0, 1]$ such that $\sum_{i=1}^{t} w_{t_0+i} \geq \alpha$,

the scheme recovers from the compromise in time $t_0 + \beta t$ where recovery occurs if $\exists\ j \in [k]$ and $\exists\ \widehat{T} \in [t_0 + 1, t_0 + \beta \cdot t]$ such that:

1. $\mathsf{out}_{t_0+1}, \ldots, \mathsf{out}_{\widehat{T}-1} \neq j$ (pool j has not been emptied before time $\widehat{T}$);
2. $\mathsf{out}_{\widehat{T}} = j$ (pool j is emptied at time $\widehat{T}$); and
3. (pool j has filled)

$$\sum_{\substack{t_0 < i \leq \widehat{T} \\ \mathsf{in}_i = j}} w_i \geq 1 \ .$$

6.3.4 Impossibility Result

We can show that, for general security, there exists an impossibility result. Specifically, we will show that for any $k \in \mathbb{N}$, there exists a choice of q such that any (k, q)-scheduler is not (α, β)-secure. In other words, for a suitable choice of q, one can break the scheduler to never recover from compromise for any α, β. Note that this is incomparable to the earlier impossibility result discussed in Section 6.2.2 as this assumes the existence of pools.

Theorem 6.8. *For any $k \in \mathbb{N}$, there exists $q^* = \alpha^2\beta^2$ such that a given (k, q)-scheduler is not (α, β)-secure for any $q \geq q^*$.*

Proof. The attack works as follows: we will provide α entropy, in $\alpha^2\beta$ steps. The security requirement is that recovery needs to happen within time $\alpha^2\beta^2$. Recall that recovery occurs if there is a pool that is emptied within $\alpha^2\beta^2$ which has total entropy of 1.

More formally, let I_j denote the j-th interval, of length $\alpha\beta$ starting from 0. This leads us to two cases:

- $\exists j^*$ such that no pool is emptied within I_{j^*}. Formally, there exists no time step i with $\mathsf{empty}_i \in I_{j^*}$.

Then, set $t_0 = \alpha\beta(j^* - 1) - 1$. After this state compromise, we provide a sequence of 1s of length α, which will set $T^* = \alpha$ and expect recovery in time $t_0 + \beta T^* = \alpha\beta j^* - 1$, which is still inside the interval I_{j^*}. However, we assumed no pool is emptied within I_{j^*}, so no recovery can be possible.

- $\forall j, \exists i$ such that $\mathsf{empty}_i \in I_j$, meaning at least one pool is emptied within all α intervals I_j.

  Set $t_0 = 0$. Then, for $j = 1, \ldots, \alpha$, pick *one* ℓ such that $\mathsf{empty}_\ell \in I_j$. Set, w_ℓ to be $1-\epsilon$ for some arbitrarily small ϵ (remaining weights are 0). At the end of this process, the adversary has provided almost α entropy, but there is no recovery, as all of these entropies are completely wasted. By making ϵ arbitrarily small, the result follows.

□

The consequence of this impossibility result is the following: there exists input weight sequence $\boldsymbol{w}$ such that, irrespective of the number of pools, we can inject entropy at a slow rate, such that no scheduler is general-secure. It also implies that we need to make some relaxations to achieve usable security results.

6.4 Reboot Secure Schedulers

The first relaxation corresponds to the situation when the system is just rebooted, i.e., we are at $t_0 = 0$. We will call this as the "reboot security" of a scheduler. This corresponds to the situation when you just turn on the computer. For this case, we can have a much simpler and better RNG, having only one pool. Like Fortuna, this pool is emptied every β^i steps for gradually increasing values of $i = 0, 1, 2, \ldots$, where β is a small integer (Windows 10 uses $\beta = 3$).

Definition 6.9 (Reboot Security of Scheduler)**.** A (k, q)-scheduler is (α, β)-reboot-secure if for $t_0 = 0$, $\forall t$ such that $t_1 = t_0 + \beta \cdot t \leq q$, and $\forall$ weights $w_1, \ldots, w_q \in [0, 1]$ such that $\sum_{i=1}^{t} w_{t_0+i} \geq \alpha$

the scheme recovers from the compromise in time $t_0 + \beta t$, where the definition of recovery is as defined in Definition 6.7.

The composition of such a reboot-secure scheduler with our "not-premature-next" PRNGs will trivially yield a "premature-next" boot PRNG, i.e., the PRNG that is used at the time when the system is booting up.

We start with a lower bound on reboot-security, irrespective of the number of pools k.

Theorem 6.10. *For a (k, q)-scheduler to be (α, β)-reboot secure, $\alpha \geq \lfloor \log_\beta(q) - \log\log q \rfloor - 1$ (i.e., $q \leq \alpha\beta^\alpha$)*

For simplicity let us assume that $q = \alpha\beta^{\ell+1}$, for some $\ell > 0$. Then, divide the time from $\alpha + 1$ to q into intervals of the following form: $(\alpha\beta^{i-1}, \alpha\beta^i]$ for $i = 1$ to $\ell + 1$. We have the following claim:

Claim 6.11. For any (α, β)-reboot secure scheduler with corresponding emptying sequence $\text{empty}_1, \ldots, \text{empty}_q$ and any $i \in [\ell]$, there must exist a t such that $\text{empty}_t \in (\alpha\beta^i, \alpha\beta^{i+1}]$. (In other words, there must be a pool that is first emptied after roughly β^i steps for every i.)

Proof. We prove this by induction. Define t to be the time within which the adversary provides α entropy, i.e.,

$$\sum_{i=1}^{t} w_i \geq \alpha$$

where these w_i are adversarially chosen. Since $w_i \leq 1$, we get that $t \geq \alpha$.

Let us assume to the contrary that there is no emptying in the interval $(\alpha, \alpha\beta]$. Now, if adversary chooses $t = \alpha$. Then, this scheme would never recover as there is no empty in the interval $(\alpha, \alpha\beta]$

Now, let us assume that there is an empty in intervals, $(\alpha, \alpha\beta], (\alpha\beta, \alpha\beta^2], \ldots, (\alpha\beta^{i-1}, \alpha\beta^i]$. We will now show that there needs to be an empty in the interval $(\alpha\beta^i, \alpha\beta^{i+1}]$. To this end, assume to the contrary. Now, note that the adversary can provide the entropy in such a way that every

empty in the preceding intervals empties out $1 - \epsilon$, without recovering. This is similar to the attack detailed in the proof of Theorem 6.8. Further, if $t = \alpha\beta^i$, the scheme has not recovered in time 1 to t and because it has no empty in $(\alpha\beta^i, \alpha\beta^{i+1}]$ it can never hope to recover in time either. Therefore, there is an empty in the interval $(\alpha\beta^i, \alpha\beta^{i+1}]$. □

Proof of Theorem 6.10. From Claim 6.11, we get that there are at least $\lfloor \log_\beta(q/\alpha) \rfloor$ distinct empties, and there needs to be entropy of 1 emptied in each of these empties. By Pigeonhole Principle, we will need $\alpha \geq \lfloor \log_\beta(q/\alpha) \rfloor$ to have any hope of recovery, which implies $\alpha \geq \lfloor \log_\beta(q) - \log\log q \rfloor - 1$. □

We now give a scheme that nearly matches the lower bound. This scheme uses the same strategy as Windows 10's "Root RNG" which is used at system startup [Fer19].

Construction 16 (Reboot Scheme)**.** The scheme has $k = 1$. $\mathsf{in}_i = 0$ for $i = 1, \ldots, q$.

$$\mathsf{out}_i = \begin{cases} 0 & \textbf{if } i = \beta^j \\ \\ \perp & \textbf{else} \end{cases}$$

In other words, $\forall i \in [\beta^{j-1}, \beta^j)$, empty at time β^j.

Theorem 6.12. *Construction 16 is (α, β)-reboot secure for $q = \alpha\beta^\alpha$ (i.e., $\alpha \approx \log_\beta q - \log\log q$).*

Proof. Define t to be the time within which the adversary provides α entropy, i.e.,

$$\sum_{i=1}^{t} w_i \geq \alpha$$

where these w_i are adversarially chosen. It is clear that $t \geq \alpha$, as we need at least α steps to provide α entropy when $w_i \in [0, 1]$.

Let i be such that $\alpha \in (\beta^{i-1}, \beta^i]$. Now, it is clear that if $t = \alpha$, then the empty at β^i will ensure recovery from compromise. We can induct similar to the proof of Claim 6.11 to get that if

$t \in [\beta^{\ell-1}, \beta^{\ell})$ for some $\ell \geq i$, then there $\exists j \in [\beta^{\ell-1}, \beta^{\ell})$ such that $w_j = 1$ (or possibly a set of such j's which sum up to 1), which is emptied at β^i, thus recovering from compromise. Specifically, if we have $t \in [\beta^{\ell-1}, \beta^{\ell})$, then at each of the preceding $\ell - 1$ intervals (each with an empty), $\mathcal{A}$ provides $1-\epsilon$ entropy, for some arbitrarily small ϵ. This gives a total of almost $\ell-1$ entropy across these intervals. Therefore, it follows that the remainder of $\alpha - \ell + 1 > 1$ needs to be provided between $w_{\beta^{\ell-1}}$ and w_t to hit α and all of these are emptied at β^{ℓ}, recovering from compromise. □

6.5 Repeat Secure Schedulers

A general secure scheme is a stronger model of security than the reboot model. This follows because the value of t_0 is also the choice of the adversary, in addition to the choice of t. However, the impossibility result from Theorem 6.8 imply a need for relaxation.

Round-Robin Schedulers. Simple round-robin schedulers achieve very good $\alpha \approx \log_\beta(q)$ for the special cases when all of the w_t are equal to some (unknown, adversarially chosen) value w, i.e.,

$$w_1 = w_2 = \ldots = w_q = w$$

and setting the number of pools $k \approx \log_\beta(q)$ (so 1 or 2 pools are too little). β is a smaller integer usually 2 or 3 in practice, as in [FS03, DSSW17]. More formally, such schedulers simply set

$$\mathsf{in}_t = t \bmod k$$

As for out_t, this is set to $\perp$ inside one round (i.e. $t \bmod k \neq 0$). At the the of each round, when $t = k\ell$, one looks at the largest index $i \geqslant 0$ such that β^i divides ℓ. Then out empties the i-th pool:

$$\mathsf{out}_t = i$$

Remark 3. There is a marginal gain in efficiency when we empty all pools $\leq i$, instead of just the i-th pool. In other words, out is a set, rather than a single index. However, for our analysis below, we will continue to work with the assumption that a single pool is emptied. (More generally, we do not make much of an attempt to optimize the parameters that we achieve. See [DSSW17] for an optimized version of similar construction.)

k-SMOOTH SEQUENCES. Our main observation is that we can significantly extend the constant-rate analysis as follows. The idea is to allow support any constant rate within a round-robin (rather than go for a constant (but unknown) rate scheduler). This constant can change arbitrarily once the next round-robin is started. Namely, we don't have to fix the same constant for all q entropies but can change it every $k \ll q$ steps. In practice, this means that while the quality of entropy can change over time, we heuristically assume that it changes rather smoothly, and we rarely have huge jumps within a given round-robin.

Definition 6.13 (Repeating Sequences)**.** $\boldsymbol{w} = (w_1, \ldots, w_q)$ with $0 \leq w_i \leq 1$ is called k-repeating if $w_{jk+1} = w_{jk+2} = \ldots = w_{jk+k}$ for $j = 0, \ldots, t-1$ where $q = k \cdot t$

Definition 6.14 (Repeat Security of Scheduler)**.** A (k, q)-scheduler is (α, β, k)-repeat-secure if $\forall\ t_0, t$ such that $t_1 = t_0 + \beta \cdot t \leq q$, and $\forall\ k$-repeating weights $w_1, \ldots, w_q \in [0, 1]$ such that $\sum_{i=1}^{t} w_{t_0+i} \geq \alpha$ the scheme recovers from the compromise in time $t_0 + \beta t$, where the definition of recovery is as defined in Definition 6.7.

To achieve such repeating sequences, we take any standard $\boldsymbol{w} = (w_1, \ldots, w_q)$ and apply a k-flattening, as defined below.

Definition 6.15 (k-Flattening)**.** Given a sequence $\vec{w} = (w_1, \ldots, w_q)$ and a number $k \geqslant 1$, where for simplicity of notation let us assume $q = kt$, we define *k-smooth flattening* of $\vec{w}$ to be $\vec{w'} = (w'_1, \ldots, w'_q)$, where for any round-robin $j \in \{0, \ldots t-1\}$ and $i \in \{1 \ldots k\}$, we let

$$w'_{jk+i} = \min(\ w_{jk+1},\ w_{jk+2}, \ldots,\ w_{(j+1)k}\ )$$

Intuitively, we change the entropy w_j to the smallest of k surrounding entropies inside a given round-robin. Of course, $k = 1$ corresponds to $w'_t = w_t$, but we already know that 1 pool is not enough (as this would give a general scheduler for the unrestricted entropy setting). For larger k, however, the flattened values could be noticeably lower than the original. For example, if $k = 3$ and $\vec{w} = \{1, 1/2, 1/3, 1/4, 1/5, 1/6\}$, the 3-flattening of $\vec{w}$ is $\vec{w'} = \{1/3, 1/3, 1/3, 1/6, 1/6, 1/6\}$. Of course, for a constant rate $w_1 = \ldots w_q = w$, k-flattening does not change anything, which explains why our results below naturally generalize the constant-rate analysis from the work of Dodis *et al.* [DSSW17].

Jumping ahead, we will see that the Fortuna scheduler is "secure" for any (normalized) entropy sequence $\vec{w}$, with the understanding that the attacker gets "entropy credit" within a single round-robin equals to k times the *lowest* entropy value in contributes within this round.

New Result. Now, we show that while the original (α, β)-definition above cannot be achieved when applied to $\vec{w}$ itself, the analysis for constant-rate schedulers works for general entropy sequences, provided we simply apply it to k-flattening of $\vec{w}$ (where $k \approx \log_\beta q$ is the number of pools) instead of $\vec{w}$ itself! Namely, a given round only gets "credit" for the smallest entropy (times k) it contributed to any of the k pools. So we do not give the adversary credit if it wildly changes the entropy values within a given round.

We now present our construction, which is parameterized by the number of pools k and a base b. One typically takes $b = 2$ or $b = 3$, and, e.g., $k = 32$ or $k = 64$ in practice, and works for $q \leq b^k$.

Construction 17 (Smooth scheduler)**.** Consider the following $(k, q := b^k)$-scheduler for integers $b \geq 2$ and $k \geq 1$:

- $\mathsf{in}_i = i \mod k$

- $$\mathrm{out}_i = \begin{cases} \perp & \textbf{if } i \mod k \neq 0 \\ j & \textbf{if } i = k\ell \end{cases}$$

  where $j \geq 0$ is the largest j such that $\ell \mod b^j = 0$ for $i = k\ell$

We now prove that this scheduler is secure (against k-repeating sequences). For simplicity, we make little attempt to optimize the parameters. See [DSSW17] for a carefully optimized version of this result for the special case where the entropy rate is constant (i.e., the case of q-repeating weights).

Theorem 6.16. *For any integers $b \geq 2$ and $k \geq 1$, Construction 17 is (α, β, k)-repeat-secure for*

$$\alpha := 3k - 2 \approx 3 \log_b q; \quad \textit{and} \quad \beta := 2b\left(1 + \frac{k}{\alpha}\right) \approx \frac{8b}{3} = \frac{8}{3} \cdot q^{1/k}$$

In particular, for $k = \log_b q$ and $q \geqslant b^2$, we have $\alpha \leqslant 3 \log_b q$ and $\beta \leqslant 3b$.

Notice, this result explains how the recovery factor β shrinks very quickly as we increase the number of pools k, starting with (roughly) q all the way down to being a constant. In particular, β becomes constant once the number of pools becomes logarithmic in q.

Moreover, up to constant factors in α and β (which, again, we do not attempt to optimize), Theorem 6.16 is tight. In particular, [DSSW17, Proposition 1] proved that even in the "constant-rate" case of q-repeating weights, no scheduler can be (α, β)-secure with $\alpha\beta \leq \log_e q - \log_e \log_e q - 1$. And our scheduler matches this bound (up to a constant factor) when $b = O(1)$ and $k = O(\log q)$.

Proof of Theorem 6.16. Let $w_1, \ldots, w_q$ be k-repeating. Let t_0 and t be such that (1) $t_0 + \beta t \leq q$; and (2)

$$\sum_{i=1}^{t} w_{t_0+i} \geq \alpha \,.$$

We wish to show that in this case the scheduler recovers before time $t_0 + \beta t$, i.e., that there exists a $j \in [k]$ and $\widehat{T} \in [t_0 + 1, t_0 + \beta t]$ such that (1) $\mathsf{out}_{\widehat{T}} = j$; (2) $\mathsf{out}_{t_0+1}, \ldots, \mathsf{out}_{\widehat{T}-1} \neq j$; and (3)

$$\sum_{\substack{t_0 < i \leq \widehat{T} \\ \mathsf{in}_i = j}} w_i \geq 1 \ .$$

Indeed, we take j to be minimal such that $\mathsf{out}_{t_0+1}, \ldots, \mathsf{out}_{t_0+t} \neq j$. In particular, notice that after pool j' is emptied, pool $j' + 1$ is not emptied for the next $k(b^{j'} - 1)$ steps. And, similarly, after pool $j' + 1$ is emptied, pool j' is not emptied for the next $k(b^{j'} - 1)$ steps. It follows that $b^{j-1} \leq t/k + 1$. Since the pool j' is emptied at least once in every $2kb^{j'}$ steps, it follows that we must have $\mathsf{out}_{\widehat{T}} = j$ for some $\widehat{T} - t_0 \leq 2kb^j \leq (2t + 2k)b \leq 2b(1 + k/\alpha)t$, where in the second inequality we have used the fact that $w_i \leq 1$, which implies that $t \geq \alpha$. In particular, $\widehat{T} \leq t_0 + \beta t$, as needed.

And, since the w_i are k-repeating, we must have

$$\sum_{\substack{t_0 < i \leq \widehat{T} \\ \mathsf{in}_i = j}} w_i \geq \sum_{t_0 < i \leqslant \widehat{T}} \sum_{\substack{t_0 < i \leq t_0 + t \\ \mathsf{in}_i = j}} w_i \geq \sum_{\substack{t_0' < i \leq t_0' + t' \\ \mathsf{in}_i = j}} w_i = \frac{1}{k} \cdot \sum_{t_0' < i \leq t_0' + t'} w_i \ ,$$

where $t_0' := \lceil t_0/k \rceil k \geq t_0$ and $t' := \lfloor t/k \rfloor k \leq t$. And, since $w_i \leq 1$, we trivially have that

$$\sum_{t_0' < i \leq t_0' + t'} w_i \geq \sum_{t_0 < i \leq t_0 + t} w_i - 2k + 2 \geq \alpha - 2k + 2 \ .$$

Therefore,

$$\sum_{\substack{t_0 < i \leq t_0 + t \\ \mathsf{in}_i = j}} w_i \geq \frac{\alpha}{k} - 2 + 2/k \geq 1 \ ,$$

as needed. □

Part II

Small-Box Cryptography

7 | Small-Box Cryptography

This chapter is based on joint work with Yevgeniy Dodis and Daniel Wichs that appeared in ITCS 2022 [DKW22a]. Passages are taken verbatim from this work. This work considers the new theoretical approach called as "small-box cryptography". This is to be contrasted with traditional cryptography which we dub as "big-box cryptography". The underpinning of "big-box cryptography" is the approach of reducing security to some assumption.

Indeed, we begin this chapter by applying such a "big-box" to build a construction of a PRG in Section 7.1. We then apply our small-box framework to build a PRG in Section 7.2. The construction matches the construction from the previous section. We finally conclude this chapter by applying the small-box framework to substitution-permutation networks (SPNs) in Section 7.3.

7.1 Applying Big-Box Cryptography to PRGs

In this section, we present our construction of a pseudorandom generator. We then prove its security under the eXact Linear Parity with Noise (XLPN) assumption. The construction, by itself, may not be the best PRG construction from this assumption, as it relies on large public parameters, which is unnecessary if one's goal to build a "big-box" PRG from XLPN. Of course, our point is to explicitly build and analyze cryptographic primitives from a "small" (but still polynomial size) S-box, which naturally mandates seemingly large parameters when viewed from the big-box perspective. Hence, the main purpose of our PRG construction is to introduce the small-box

framework, before we look at the more complicated example of block ciphers in Section 7.3. In particular, unlike the case of block ciphers, the example will be simple enough that we can directly apply the "big-box" analysis to it (in the common reference string model, modeling our S-box).

7.1.1 Syntax and Security of PRG

A PRG is a primitive that is often used to produce random-looking string from a short, randomly chosen seed.

Definition 7.1 (Pseudorandom Generator)**.** Let $n \in \mathbb{N}$ be the security parameter. Then, an efficiently computable function $G : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ for $\ell(n) > n$ is an (T, ϵ)-secure PRG if for all adversaries $\mathcal{A}$ running in time T, the following holds:

$$\left|\Pr[]s \leftarrow U_n\left[\mathcal{A}(G(s)) = 1\right] - \Pr[]R \leftarrow U_{\ell(n)}\left[\mathcal{A}(R) = 1\right]\right| \leq \epsilon$$

7.1.2 Our Construction

Recall, the goal of small-box cryptography is to analyze the direct construction of various primitives from "small" (constant- or polynomial-, but not exponential-) sized S-boxes. In the case of a PRG, it is natural to think of such an S-box as a Boolean function f modeled as a random function in the analysis. This is without loss of generality, as any non-Boolean S-box $f' : \{0,1\}^a \to \{0,1\}^b$ is equivalent to a Boolean S-box $f : \{0,1\}^{a+\log b} \to \{0,1\}$, where $f(x\|i)$ represents the i-th output bit of $f'(x)$. Further, it will be convenient for the notation to write the domain of this Boolean function as $\{0,1\}^{n+\log \ell}$, where ℓ is the desired output of our PRG, and n is the "small" leftover part. E.g., when $n = 8$ and $\ell = 256$, we get (still "small") 16-to-1 S-box.

For our "big-box" analysis, it will also be convenient to define a truth-table matrix for f as an $\ell \times N$ matrix $\mathbf{M}$, and think of this matrix as public parameters (or common random string, crs) of our PRG construction:

$$\mathbf{M} = \begin{pmatrix} f(1 \parallel 0) & \dots & f(N \parallel 0) \\ f(1 \parallel 1) & \dots & f(N \parallel 1) \\ \vdots & \ddots & \vdots \\ f(1 \parallel \ell - 1) & \dots & f(N \parallel \ell - 1) \end{pmatrix}$$

where $N = 2^n$.

Let $\mathcal{F} = \{f : \{0,1\}^{n+\log \ell} \to \{0,1\}\}$ be the set of all "S-box" functions f above. We now define a family of PRGs $\mathcal{G} = \{\tilde{G}_f : \{0,1\}^{nc} \to \{0,1\}^{\ell} \mid f \leftarrow \mathcal{F}\}$, which takes an additional "hardness" parameter c, and will expand a cn-bit input $x = (x_1, \dots, x_c)$ into an ℓ-bit output y as follows:

$$y = \tilde{G}_f(x_1, \dots, x_c) = \begin{pmatrix} f(x_1 \parallel 0) \oplus f(x_2 \parallel 0) \oplus \dots \oplus f(x_c \parallel 0) \\ f(x_1 \parallel 1) \oplus f(x_2 \parallel 1) \oplus \dots \oplus f(x_c \parallel 1) \\ \vdots \\ f(x_1 \parallel \ell - 1) \oplus f(x_2 \parallel \ell - 1) \oplus \dots \oplus f(x_c \parallel \ell - 1) \end{pmatrix}$$

Note on parameters. We need $\ell \geq nc + 1$ in order to ensure that our PRG is expanding, which lower bounds the domain length of the S-box by $(n + \log(nc + 1)) = O(\log c)$, if we think of $n = O(\log c)$. This is still a pretty good trade-off. Indeed, in both of our big- and small-box analyses (done in Sections 7.1.3 and 7.2), c will be the "security" parameter of the construction. So our security will scale — under appropriate hardness assumptions — exponentially in c. While the bit-size of the S-box input has only logarithmic dependence on the security parameter c. In particular, while the overall size of the S-box $\ell \cdot 2^n \approx c \cdot (n2^n)$ is noticeably greater than the PRG input size $c \cdot (n + \log \ell) \approx c \cdot (n + \log c)$, it is still polynomial in the security parameter c (assuming $n = O(\log c)$), and can be read by the attacker in its entirety.

7.1.3 Big-Box Analysis of $\tilde{G}$

In this section, we will undertake a big-box analysis of $\tilde{G}$ by proving its security from well-studied assumption, a variant of the LPN problem. The variant we consider is called the Exact LPN problem. This was first proposed and employed in proof of security by Jain *et al.* [JKPT12]. Much like the original LPN problem, the XLPN problem has a search and a decisional variant. It has been shown that the search variant of this problem is equivalent to the search version of the original LPN problem. Additionally, the hardness of the decisional XLPN problem is polynomially related to the search LPN problem.

Definition 7.2 (Decisional Exact LPN (XLPN) Assumption)**.** For $0 < \tau < \frac{1}{2}$, $q, m \in \mathbb{N}$, the (q, m)-XLPN_τ problem is (T, ϵ)-hard if for every adversary $\mathcal{A}$ running in time T, the following holds:

$$\left|\Pr[]\mathbf{s}, \mathbf{A}, \mathbf{x}\left[\mathcal{A}(\mathbf{A}, \mathbf{A}^\top\mathbf{s} \oplus \mathbf{x}) = 1\right] - \Pr[]\mathbf{A}, \mathbf{y}\left[\mathcal{A}(\mathbf{A}, \mathbf{y}) = 1\right]\right| \leq \epsilon$$

where $\mathbf{s} \leftarrow \mathbb{Z}_2^m$, $\mathbf{A} \leftarrow \mathbb{Z}_2^{m\times q}$, $\mathbf{x} \leftarrow \mathbb{Z}_{2,c}^q$ and $\mathbf{y} \leftarrow \mathbb{Z}_2^q$. Here, $\mathbb{Z}_{2,c}^q$ is the uniform distribution of q dimension binary vectors of weight $c = \tau \cdot q$.

To this end, we will prove the following theorem:

Theorem 7.3. *Under the $(q = N, m = N - \ell)$-XLPN_τ assumption, the family of PRGs $\mathcal{G} = \{\tilde{G}_f : \{0,1\}^{nc} \to \{0,1\}^\ell | f \leftarrow \mathcal{F}\}$ is secure and provided $c = 2^n \cdot \tau$ and $\ell \geq nc + 1$, for $0 < \tau < \frac{1}{2}$.*

Discussion on parameters. Note that the length doubling PRG has an error-rate of $1/O(\log n)$, which is worse than a constant, but much better than $1/O(\sqrt{N})$ needed for public-key encryption. Finally, by suitably setting the parameters, we get the following result:

Corollary 7.4. *For any polynomial N, let $\ell = N/2$ and $c = \ell/(2\log N) = N/(4\log N)$. Then, there exists a family of length-doubling PRG under the $(N, N/2)$-XLPN_τ assumption where $\tau = 1/O(\log N)$.*

Before we look at the proof, we discuss some instructive intuitions for the proof. Recall that in the PRG security game, the adversary $\mathcal{A}$ either receives $\tilde{G}(\boldsymbol{X})$ for $\boldsymbol{X} \leftarrow \{0,1\}^{nc}$ or $\mathbf{y} \leftarrow \{0,1\}^{\ell}$. To break this game, $\mathcal{A}$ would have to identify c values $x_1, \ldots, x_c$ that evaluates to the output that it has received, and in this setting $\mathbf{y}$ is a set of ℓ parity check equations.

In other words, if $\mathcal{A}$ finds a vector $\mathbf{x} \in \mathbb{Z}_2^N$ such that $wt(\mathbf{x}) = c$ and $\mathbf{Mx} = \mathbf{y}$, then with high probability, $\mathcal{A}$ received the real value and not the random value.

With this insight, it is useful to view this problem via the context of linear binary codes. In such a case, $\mathbf{M}$ can be considered as a parity check matrix and $\mathbf{y}$ is the syndrome of $\mathbf{x}$. However, this only works if $\mathbf{M}$ is of full row rank. Recall that a matrix $\mathbf{M}$ has a full row rank. if each of the rows of the matrix is linearly independent. Fortunately, we know that with overwhelming probability, a randomly sampled binary matrix has full rank.

In other words, given a random parity-check matrix $\mathbf{M}$ of size $\ell \times N$, we need to decode a random error vector $\mathbf{x}$, from the ℓ parity check equations, i.e., $\mathbf{Mx} = \mathbf{y}$, such that $wt(\mathbf{x}) = c$. Further, we get that $\binom{N}{c} < 2^{\ell} \implies c \log N < \ell < N$

Finally, given a parity-check matrix $\mathbf{M}$, one can efficiently calculate a corresponding generator matrix $\mathbf{A}$. Note that $\mathbf{A} \in \mathbb{Z}_2^{(N-\ell)\times N}$ and $\mathbf{MA}^{\top} = \mathbf{0}$, by definition.

Proof. With the above intuition, we can prove the hardness amplification result through a sequence of hybrids, and reducing the problem to a variant of the LPN problem. In the proof we denote the uniform distribution of binary vectors of length N and weight c by $\mathbb{Z}_{2,c}^N$.

Hybrid H_0. $\mathcal{A}$ receives $\mathbf{Mx}$ for $\mathbf{x} \leftarrow \mathbb{Z}_{2,c}^N$ and $\mathbf{M} \leftarrow \mathbb{Z}_2^{\ell \times N}$.

Hybrid H_1. $\mathcal{A}$ receives $\mathbf{Mx} \oplus \mathbf{MA}^{\top}\mathbf{s}$ where $\mathbf{A}$ is the generator matrix corresponding to the parity check matrix $\mathbf{M} \leftarrow \mathbb{Z}_2^{\ell \times N}$. $\mathbf{A} \in \mathbb{Z}_2^{(N-\ell)\times N}$, $\mathbf{s} \leftarrow \mathbb{Z}_2^{N-\ell}$, and $\mathbf{x} \leftarrow \mathbb{Z}_2^N$ with $wt(\mathbf{x}) = c$

Note that Hybrids H_0 and H_1 are identically distributed because of the property that $\mathbf{MA}^{\top} = \mathbf{0}$

Hybrid H_2. $\mathcal{A}$ receives $\mathbf{Mx} \oplus \mathbf{MA}^{\top}\mathbf{s}$ where $\mathbf{M}$ is the parity check matrix corresponding to the generator matrix $\mathbf{A} \leftarrow \mathbb{Z}_2^{(N-\ell)\times N}$ $\mathbf{s} \leftarrow \mathbb{Z}_2^{N-\ell}$, and $\mathbf{x} \leftarrow \mathbb{Z}_2^N$ with $wt(\mathbf{x}) = c$

Note that the difference between Hybrids H_1 and H_2 only lies in the order of sampling $\mathbf{M}, \mathbf{A}$. In H_1, we sample $\mathbf{M}$ and then compute $\mathbf{A}$, while in H_2 we do the opposite.

Hybrid H_3. $\mathcal{A}$ receives $\mathbf{Me}$ where $\mathbf{M}$ is the parity check matrix corresponding to the generator matrix $\mathbf{A} \leftarrow \mathbb{Z}_2^{(N-\ell)\times N}$ and $\mathbf{e} \leftarrow \mathbb{Z}_2^N$.

Claim 7.5. If $(N, m = N - \ell)$-XLPN$_\tau$ is (t, ϵ)-hard, then the distinguishing advantage between H_2 and H_3 for any PPT adversary $\mathcal{A}$ is at most ϵ provided $c = N \cdot \tau$

Proof. Let us assume that there is $\mathcal{A}_2$ that can distinguish between H_2 and H_3. We will construct $\mathcal{A}_1$ that uses $\mathcal{A}_2$ to win the ranked LPN game.

Challenger samples $\mathbf{A} \leftarrow \mathbb{Z}_2^{(N-\ell)\times N}$, $\mathbf{s} \leftarrow \mathbb{Z}_2^{N-\ell}$, and $\mathbf{x} \leftarrow \mathbb{Z}_2^N$ with $wt(\mathbf{x}) = c$. It then sets $\mathbf{e}_0 = \mathbf{A}^\top \mathbf{s} \oplus \mathbf{x}$ and $\mathbf{e}_1 \leftarrow \mathbb{Z}_2^N$. It tosses a bit and sends to $\mathcal{A}_1$, $(\mathbf{A}, \mathbf{e} = \mathbf{e}_b)$. $\mathcal{A}_1$ then generates the corresponding PCM $\mathbf{M}$ for $\mathbf{A}$ and runs $\mathcal{A}_2$ on $\mathbf{Me}$. It is easy to verify that if $b = 0$, $\mathcal{A}_1$ simulates perfectly H_2 and if $b = 1$, it simulates H_3 perfectly. $\mathcal{A}_1$ merely forwards $\mathcal{A}_2$'s guess as its own. This concludes the proof. □

Hybrid H_4. $\mathcal{A}$ receives $\mathbf{Me}$ where $\mathbf{M} \leftarrow \mathbb{Z}_2^{\ell\times N}$ and $\mathbf{e} \leftarrow \mathbb{Z}_2^N$.

Note that the difference between hybrids H_3 and H_4 is again the order of sampling. In the former, $\mathbf{A}$ is sampled and then $\mathbf{M}$ is computed, whereas in the latter $\mathbf{M}$ is directly sampled.

Hybrid H_5. $\mathcal{A}$ receives $\mathbf{y} \leftarrow \mathbb{Z}_2^\ell$

Hybrids H_4, H_5 are identically distributed and therefore are statistically indistinguishable.

□

7.2 Applying Small-Box Cryptography to PRGs

In the previous section, we presented the construction of a PRG, using an idealized primitive f, and proved its security under the XLPN assumption. In this section, we arrive at the same

construction, but by religiously following the small-box framework. Recall, our recipe for small-box cryptography consists of two steps — the *construction step* and then the *analysis step*, each of which consists of several small steps. We detail each below.

7.2.1 Construction Step

The construction step of small-box cryptography consists of two smaller sub-steps: *domain extension* and *hardness amplification*. Although both of these steps are primitive-specific (e.g., different from PRGs and block ciphers), they are largely syntactic and require little-to-no technical expertise.

Domain Extension Step. Normally, the ideal object (S-box) gives a direct construction of the given primitive, but for "tiny" input/output domain. For example, in the PRG case the S-box $f : \{0,1\}^{n+\log \ell} \to \{0,1\}$ is a trivial "PRG" from $(n + \log \ell)$ bits to 1 bit. Of course, being non-expanding, this is not interesting in terms of functionality, but it will be obviously "secure" when we think of n as "big" and f as a "big" random oracle in subsequent sections.

To make the primitive interesting in terms of functionality even in the small-box world, the purpose of the domain extension step is to amplify the length of either the input, the output, or both to be large even in the "small" box world. In the case of PRG, the interesting parameter is the desired PRG output length ℓ, which we think as "big".[1] So our goal here is to extend the output domain from $\{0,1\}$ to $\{0,1\}^{\ell}$.

In the big-box world, one would amplify the output size by a factor of ℓ by expanding the PRG seed length by a factor of ℓ and concatenating the ℓ outputs of the base PRG. Here we do almost the same thing, except we don't need to pay in the seed length, and use our idealized modeling of our base PRG f as a random oracle rather than a "mere" PRG. This is consistent with the design intuition that a good S-box has all the idealized properties one would need for the construction

[1] This explains our strange-looking choice of notation to denote the input length of our S-box as $(n+\log \ell)$ rather than just ℓ. Of course, this is just matter of convenience of notation: if the S-box size was n', we would have to subtract $\log \ell$ from it, and instead assume $n' = \log \ell + n$ for a new parameter n.

to work. Namely, we can construct the range-extended PRG G as follows: $G : \{0,1\}^n \to \{0,1\}^\ell$:

$$G(x) = (f(x \,\|\, 0), \ldots, f(x \,\|\, \ell - 1)) \tag{7.1}$$

where $\|$ denotes concatenation. Intuitively, we simply "waste" $\log \ell$ bits of the seed to enumerate over the ℓ desired output bits.

Hardness Amplification Step. As we can see, the improved functionality — in this case, output size — came at the expense of decreased security (which is, of course, expected). For the PRG example above, the seed length was $(n + \log \ell)$ bits, but now is only n bits, which means it is definitely easier to break (we will formalize this quantitatively in Section 7.2.2).

The goal of the hardness amplification step is to amplify security — not just to the level we started from — but hopefully well beyond, so that we can afford to make n "small" and still have good looking security bound (this is somewhat subtle, and will be explained in the analysis step in Section 7.2.2). The hardness amplification step is usually parameterized by the hardness parameter c, which we can also think of as a security parameter of our final construction. For the case of PRGs, the standard hardness amplification is simply the bit-wise XOR operation, applied to c independent copies of our (already "domain-extended") PRG. Intuitively, while each individual PRG might only be slightly secure, by XOR-ing c independent copies the potential biases of the final PRG decay exponentially in c. This was formally analyzed in the computational setting by Dodis *et al.* [DIJK09] and in the information-theoretic setting by Maurer *et al.* [MPR07].

With this in mind, we can define the following PRG $\tilde{G} : \{0,1\}^{nc} \to \{0,1\}^\ell$:

$$\tilde{G}(x_1, \ldots, x_c) = G(x_1) \oplus \ldots \oplus G(x_c)$$

This PRG can also be rewritten as follows, if we unwrap the definition of G from Equation (7.1):

$$\tilde{G}(x_1, \ldots, x_c) = \begin{pmatrix} f(x_1 \,\|\, 0) \oplus f(x_2 \,\|\, 0) \oplus \ldots \oplus f(x_c \,\|\, 0) \\ f(x_1 \,\|\, 1) \oplus f(x_2 \,\|\, 1) \oplus \ldots \oplus f(x_c \,\|\, 1) \\ \vdots \\ f(x_1 \,\|\, \ell - 1) \oplus f(x_2 \,\|\, \ell - 1) \oplus \ldots \oplus f(x_c \,\|\, \ell - 1) \end{pmatrix} \tag{7.2}$$

This is the same construction as the one in Section 7.1.2, but now obtained using two relatively syntactic steps. In each step, we intuitively think of f as a "big" random oracle to justify the soundness of this step (and we formalize this below), but the actual construction makes sense even in the "small-box" world! This dichotomy will be the point of the analysis step we present in the next section.

7.2.2 Analysis Step

On a high-level, the analysis step of small-box cryptography will consist of two components. The first component is *provable*, typically information-theoretically. It involves the analysis of the security of the final object ($\tilde{G}$, in the case of PRG, or SPN cipher in the case of block ciphers) in the corresponding idealized model for the building block f (random oracle model, in the case of PRG, and random permutation model in the case of SPNs). The proof will critically use the assumption that the size of f is larger than the running time T of the attacker $\mathcal{A}$ so that $\mathcal{A}$ cannot query f on all inputs. However, the final security bound one gets will be "syntactically meaningful" even in the small-box world, when the size of f becomes polynomial. Then the second component of the analysis will involve a new type of conjecture, which we term *Big-to-Small conjecture*, which was never considered prior to this work, and which allows one to get good exact security bounds for the final construction in the small-box world. We detail these below for the simple case of PRGs.

Idealized Big-box Proof. Here we are arguing the security of our final PRG $\tilde{G}$ in the random oracle model for the S-box f. Normally, one would try to do it modularly, by separately analyzing the domain extension step, followed by the hardness amplification step. Indeed, this is how we will do the analysis in the case of SPNs, where a direct analysis of the entire construction appears extremely cumbersome. Here, however, the PRG construction is so simple, that we do a direct proof for the security of $\tilde{G}$ in the random oracle model for f.

Recall that in the basic PRG security game, an adversary has to distinguish between $\tilde{G}(x)$ and a random ℓ-bit string, for a random seed $x = (x_1, \ldots, x_c)$, by making at most q queried to the random oracle f. We obtain the following simple lemma:

Lemma 7.6. *Let $f : \{0,1\}^{n+\log \ell} \to \{0,1\}$ be modeled as a random oracle. Then, $\tilde{G} : \{0,1\}^{nc} \to \{0,1\}^{\ell}$ is $(q/N)^c$-secure PRG where $N = 2^n$, and q is the number of oracle queries made to f.*

Proof. Let us define the variable q_j to be the number of calls to f of the form $f(\cdot, j)$ for $j = 0, \ldots, \ell - 1$. Let $x_1, \ldots, x_c$ be n-bit strings, randomly sampled as the seeds. Now, define an event Bad_j as the event that a PPT attacker $\mathcal{A}$ invoked $f(x_1, j), \ldots, f(x_c, j)$. Now, note that the the probability that $\mathcal{A}$ invoked exactly one of these seeds with j is at most $q_j/2^n$. Therefore, $\Pr[[]Bad_j] \leq (q_j/2^n)^c$.

Define by $\mathcal{E}$ the event that any of $Bad_1, \ldots, Bad_{\ell-1}$ occurred. Then, we know that

$$\Pr[[]\mathcal{E}] = \sum_{j=0}^{\ell-1} \Pr[[]Bad_j] = \frac{1}{N^c} \sum_{j=0}^{\ell-1} q_j^c \leq \left(\frac{q}{N}\right)^c$$

Now, note that if $\mathcal{E}$ did not happen, then the adversary has no distinguishing advantage between real or random. Therefore, the distinguishing advantage of $\mathcal{A}$ in the PRG game is $(q/N)^c$. □

Removing the dependence on q in ϵ. We need one other syntactic, but extremely important step. For reasons to be clear when we move to the Big-to-small conjecture, we cannot afford to have a dependence on a number of oracle queries q in our security bound for ϵ. Instead, we will re-write our bound, but in a way that pushed the dependence on q into the lower bound for the S-

box input parameter n. Concretely, if we (temporarily) assume that $n \geqslant 10 \log q$ (or, equivalently, $q \leqslant 2^{n/10}$), then $\epsilon(n) \leqslant 2^{-0.9nc} = N^{-0.9c}$.

Finally, we will now no longer assume that the attacker $\mathcal{A}$ is computationally unbounded, but instead upper bound its running time by some parameter $T \geqslant q$, and say that our PRG is (T, ε)-secure if no such attacker can break it with an advantage more than ϵ. With this change, we get the following restatement on our bound in Lemma 7.6 which will be convenient for our Big-to-small conjecture.

Theorem 7.7. *If $n \geqslant 10 \log T$ and $f : \{0,1\}^{n+\log \ell} \to \{0,1\}$ is modeled as a random oracle, then $\tilde{G} : \{0,1\}^{nc} \to \{0,1\}^{\ell}$ given in Equation (7.2) is a $(T, N^{-0.9c})$-secure PRG, where $N = 2^n$.*

Big-to-Small Conjecture. Our analysis in the sections thus far have assumed that n is sufficiently large, i.e., "big n". Formally, Theorem 7.7 assumed that $n > 10 \log T$. However, the construction of $\tilde{G}$ is interesting even when n is much smaller. Indeed, we only need $cn < \ell$ to get a meaningful expansion. Moreover, even the final security bound $N^{-0.9c}$ is pretty good (while not established, of course!) for quire reasonable values of n and c. For example, setting $c = n = 8$ and $\ell = 128$, we get a PRG with seed length $cn = 64$, output length $\ell = 128$, and conjectured security $(2^{-64})^{-0.9} \approx 2^{-57}$, from a reasonably small Boolean S-box on 15 bits (or, equivalently, a more "balanced" S-box from 12-to-8 bits, which is quite reasonable to build). This would be fantastic, if true!

Of course, such security makes no sense, as it does not depend on the running time T of the distinguisher. Indeed, we could have replaced $n \geqslant 10 \log T$ with the bound $n \geqslant 1000000 \log T$, and basically get optimal security $\approx 2^{-nc}$ using a cn-bit seed, without doing any work. Nevertheless, we conjecture that bounds such as the one in Theorem 7.7 are hopefully meaningful for real-world security of the corresponding ciphers, provided one also includes some term corresponding to "brute-force attacks" running in time T. For example, the best generic (non-uniform) attacks against PRGs with cn-bit key [DTT10] have an advantage roughly $T/N^{c/2}$ using non-uniform

attackers using time and space T.

A particularly strong Big-to-small conjecture[2] would then state that the best way to attack constructions of the type we present is either by doing a brute-force search with advantage $T/N^{c/2}$ ignoring the fine-grained structure of our PRG, or we could have a generic attack on the structure of our PRG, ignoring its key size. And since with such a strong conjecture we have $T/N^{c/2} \gg N^{-0.9c}$, we are effectively saying that the brute-force attack is the best we can do for our cipher.

Of course, we could make weaker conjectures, and perhaps invest more time in the crypt-analysis of the resulting cipher. But the "mega-conjecture" of our approach is as follows:

Big-to-Small (Meta-)Conjecture: *If the idealized big-box analysis shows $(T, N^{-\alpha c})$-hardness when $n > a \log T$ (for some $a > 1$ and $\alpha < 1$) for the c-time iterated construction of a given primitive, then the construction is also $(T, N^{-\alpha c} + \epsilon(T))$-secure for any $n \geqslant n_0$, where $n_0 = n_0(a, \alpha) \ll \log T$ is a constant, and $\epsilon(T)$ accounts for a term involving a brute-force search component in time T.*

Conjecture 7.8 (Big-to-Small Conjecture; Informal)**.** *Assume a PRG G' of seed length $\ell(n)$ is $(T, \epsilon'(n))$-secure, where $\varepsilon'(n) > T/2^{\ell(n)}$, when using ideal building component of length $n \geqslant a \log T$ (for some $a > 1$). Then, for some constant $n_0 = n_0(a)$, the "scaled down" version of G' of seed length $\ell(n_0)$ using building block f of size $n \geqslant n_0$ is still $(T, O(\varepsilon'(n))$-secure.*

We defer a more precise discussion on such a conjecture, its practicality, and its impact after a similar analysis of SPNs in Section 7.3.3, as this is our most interesting case.

We note, however, that we would not be surprised that such a strong conjecture could be false in its generality. For example, analogous conjecture is clear false for related unpredictability primitives, such as one-way functions (OWF) constructed using direct product with independent inputs: $F(x_1, \ldots, x_w) = f(x_1), \ldots, f(x_w)$. Namely, when scaling the input length n to OWF f from security parameter to constant, we clearly make the resulting combined function

[2] Of course, we have no chance of proving such a conjecture, as it clearly implies one-way functions.

F insecure, by iterative inverting each x_i one by one. However, it currently appears that funding natural counter-examples for indistinguishability primitives (like PRGs and block ciphers) is quite non-obvious, even if one starts with artificial constructions not motivated by what is done in practice. Moreover, once the corresponding primitive is built using the natural hardness amplification step applied c times (e.g., cascade for block ciphers, or XOR for PRGs), the big-to-small conjecture becomes quite plausible. Indeed, we believe it could be true (while beyond our reach formally), at least with a weaker security term $N^{-a'c}$ for $a' < a$ (when the non-cascaded version has security N^{-a}). Further, the we would not be surprised if the brute-force component $\epsilon(T)$ could be improved by future cryptanalysis to be somewhat below the naive brute-force search.

To sum up, while many aspects of our framework are still being nailed down, we hope this work will motivate further explorations of small-box cryptography, including its promise and limitations.

7.3 Applying Small-Box Cryptography to SPNs

As our next result, we demonstrate the use of our framework to obtain concrete security bounds for SPN block ciphers.[3] In Section 7.3.1 we remind the reader of the syntax of (linear) SPNs. In Section 7.3.2 we show how we can obtain essentially the same construction by combining a "domain extension step" with the "hardness amplification" step. Namely, the former could be viewed as reduced-round SPN for which we will use the results of [CDK+18], which showed that 3-round linear SPNs achieve $O(T^2/2^n)$ security in the random permutation model (as a way to model the S-box, and under pretty mild restrictions on the linear D-box design). As stated before, a D-box is keyed, non-cryptographic permutation on wn bits. The latter step of "hardness amplification" could be viewed as cascading the cipher with independent (or correlated) keys to increase the number of rounds to get below 2^{-n} security barrier (in the "big-box" world). These

[3] Although we only apply our result to the SPN design, the discussion below is rather general, and can be applied to any r-round design E which uses some idealized building block f of (potentially small) size n.

analyses are done in Sections 7.3.3. Finally, Section 7.3.3 formalizes an appropriate "big-to-small" conjecture to go to the "small-box" world, and Section 7.3.4 brings everything together to justify Theorem 1.1 and get the concrete (conjectured) security bounds advertised in the Introduction.

7.3.1 Pseudorandom Permutations and SPNs

Pseudorandom Permutation. We now look at the security of a Pseudorandom Permutation (PRP).

Definition 7.9 (Pseudorandom Permutation)**.** Let $n \in \mathbb{N}$ be the security parameter. Then, an efficiently computable keyed-permutation $E_k : \{0,1\}^n \to \{0,1\}^n$ where $k \leftarrow \{0,1\}^s$ is an (T, ϵ)-secure PRP if for all adversaries $\mathcal{A}$ running in time T, the following holds:

$$\left| \Pr[]k \leftarrow \{0,1\}^s \left[\mathcal{A}^{E_k(\cdot)}() = 1\right] - \Pr[]P \leftarrow \mathcal{P} \left[\mathcal{A}^{P(\cdot)}() = 1\right] \right| \leq \epsilon$$

where $\mathcal{P}$ is the set of all permutations over $\{0,1\}^n$. Note that if the construction uses an ideal object, then $\mathcal{A}$ gets oracle access to this primitive as well.

Substitution-Permutation Networks. A *substitution-permutation network (SPN)* is a keyed permutation defined by the two transformations that it repeatedly invokes. The first transformation is what is called an "S"-box where one computes, block by block, a public, cryptographic permutation. The second transformation uses a keyed, non-cryptographic permutation. The repeated invocation is determined by the rounds of the SPN. In addition, the distribution of the keys for the keyed-permutation is also included in this definition, though in practice, the keys are actually derived from a single master key through a *key schedule*.

Formally, an r-round SPN taking inputs of length wn where $w \in \mathbb{N}$ is the *width* of the network, is defined by:

1. $r+1$ keyed permutations $\{\pi_i : K_i \times \{0,1\}^{wn} \to \{0,1\}^{wn}\}_{i=0}^{r}$,

2. a distribution $\mathcal{K}$ over $K_0 \times \cdots \times K_r$, and

3. a permutation $f : \{0,1\}^n \to \{0,1\}^n$.

The actual construction is as follows:

- $x_1 := \pi_0(k_0, x)$.
- For $i = 1$ to r do:
  1. $y_i := \overline{S}(x_i)$, where $\overline{S}\left(x[1] \parallel \cdots \parallel x[w]\right) \stackrel{\text{def}}{=} f(x[1]) \parallel \cdots \parallel f(x[w])$.
  2. $x_{i+1} := \pi_i(k_i, y_i)$.
- The output is x_{r+1}.

where $(k_0, \ldots, k_r) \in K_0 \times \cdots \times K_r$ are the round keys and $x \in \{0,1\}^{wn}$ is the input.

Note that if f is efficiently invertible and each π_i is efficiently invertible (given the appropriate key), then one can simply reverse the process, given the round keys, to obtain the original input x.

Linear SPNs. In practice, majority of SPNs are what we call *linear*. Such SPNs correspond to the setting where the D-Boxes (i.e., the keyed permutations π_i) are defined as follows: $\pi_i(k_i, y) = k_i + y$, where each $k_i = T_i(k)$ with T_i being a linear transformation, and k being the "main" key. A simple example of such linear SPN corresponds to the case there each T_i is the identity function, meaning the original key $k = (k_0, \ldots, k_r)$ is $(r+1)wn$-bit long, and consists of $(r+1)$ independent sub-keys of length wn each. However, we could have more compact *key schedules* $T = (T_0, \ldots, T_r)$, where the main key k will be much smaller (and each function T_i possibly expanding). Indeed, such linear SPNs were analyzed by Cogliati *et al.* [CDK+18] (see Lemma 7.10 and Lemma 7.11 below).

Figure 7.1 is a pictorial representation of a 3-round Linear SPN with unspecified linear transformations T_0, T_1, T_2, T_3.

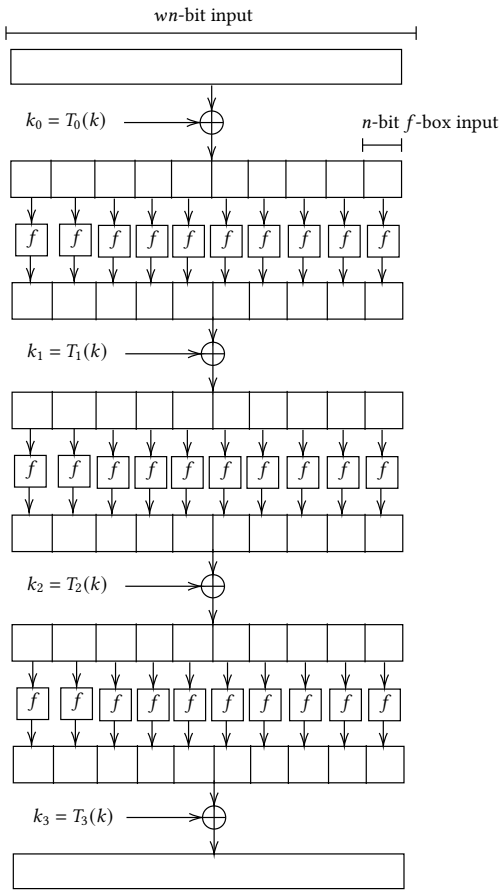

Figure 7.1: A 3-round Linear SPN with key schedule (T_0, T_1, T_2, T_3) expanding k to rounds keys (k_0, k_1, k_2, k_3), where $k_i = T_i(k)$ for $i = 0, 1, 2, 3$

7.3.2 Construction Step

In this section, we show how the defined SPN can be "syntactically" obtained through a process of two steps — domain extension and hardness amplification.

Domain Extension Step. In this step, we view the S-box as an idealized block (random permutation), and our goal is to find the minimal number of rounds r for which SPNs (with appropriately chosen linear D-boxes) are $(T, 2^{-\Omega(n)})$-secure in the random permutation model. This is exactly the question studied by [CDK+18], who showed that minimal such $r = 3$, and we will use their concrete results in Section 7.3.3.

Hardness Amplification Step. First, since we are in the big world, we imagine the size n of the "small-box" f is made large enough so that exponential in n security is meaningful. For example, one could imagine SPN ciphers with large S-boxes (say, of several hundred bits long), even though they yield block ciphers of much higher block length wn than we might need (say, thousand bits or more). Then one can ask the question if the security of such "blown up" ciphers (still with idealized f) gets significantly better when one starts to increase the number of rounds r well beyond what is needed for their minimal security, by cascading the block cipher with itself, with independently generated keys. This is exactly the question of hardness amplification of block ciphers studied by [MPR07, Tes11]; their result states that by cascading c independent, (T, ε)-secure ciphers, one still gets (T, ε')-security which decays exponentially in c: $\varepsilon' \approx \varepsilon^c$, but for our purposes any weaker exponential dependence on c (e.g., $\varepsilon' = \varepsilon^{c/100}$) will be enough to get a meaningful result, at the price of lesser efficiency. We give a more precise analysis in Section 7.3.3.

In summary, by doing this c-cascading step applied to the basic 3-round SPN predicted secure by [CDK+18] in the big-box world, we effectively obtain $3c$-round SPN, which was exactly our goal.

7.3.3 Analysis Step

Soundness of Domain Extension. As our next step, we analyze the soundness of hardness amplification in the big-box world, when we still model f as a "big" ideal object. As for the PRG case, we do it in the information-theoretic setting, where the running time of the attacker is unbounded, and only the number of oracle queries q is still bounded. Unlike the PRG case, the direct analysis of both domain extension and hardness amplification together appears extremely involved. Instead, we do it in a modular fashion, starting with the analysis of domain extension.

Fortunately for us, this question has been studied by Cogliati *et al.*[CDK+18]. They study the security of an SPN as a strong-pseudorandom permutation. Specifically, they show that a 2-round SPN is insecure with linear D-boxes but a 3-round SPN is secure, with caveats. Formally, these

are the results for the 3-Round SPN which we present here, without proof. We invite the readers to refer to the original work for a complete discussion on the two Lemmas that we will use below.

Lemma 7.10 (Security of 3-Round SPN, Corollary 1 [CDK+18])**.** *For $w > 1$, there exists a 3-round linear SPN $k_0 = k_3 = k$ for uniform $k \in \{0,1\}^{wn}$ and set $k_1 = k_2 = 0^{wn}$ which is $\epsilon(q) = O(q^2/2^n)$-secure, where q is the number of queries made by the distinguisher.*

Lemma 7.11 (Security of 3-Round SPN, Corollary 2 [CDK+18])**.** *Let $w > 1$, k' be a uniform n-bit key, and a_i for $i = 1, \ldots, w$ are distinct non-zero elements of finite field $\mathbb{F} = \mathrm{GF}(2^n)$. Then, there exists a 3-round linear SPN with $k_0[i] = k_3[i] = a_i \cdot k'$, $k_1 = k_2 = 0^{wn}$ which is $\epsilon(q) = O(q^2/2^n)$-secure.*

Lemma 7.10 deals with the minimal security of the 3-round scheme. However, one can reduce the key length from wn to n (saving a factor of w), and Lemma 7.11 shows such reduction in key length still leaves the construction almost as secure, by utilizing a more aggressive key schedule.

Provable Hardness Amplification with Independent Keys. We begin by unconditionally *proving* the hardness amplification that we need (under appropriate independence assumptions) using a beautiful hardness amplification result of Maurer, Pietrzak, and Renner [MPR07]. This is proved for a cascade of c block ciphers $E_1, \ldots, E_c$ which use both independent keys and independent ideal components f. For the case of SPNs, this means independent S-boxes with independent round keys. (We comment on how to relax this assumption later in the section.)

In the language of [MPR07], imagine we have two indistinguishable "random systems" F and H, where:

- F provides two oracles, where the first oracle is the ideal building block f of length n, and the second oracle is the (keyed) block-cipher construction E_k^f utilizing f as an oracle and using a secret key k. Denote such block cipher by $E = E_k^f$, and $F = (f, E)$. Note, both forward and backward queries to E are allowed (and the same is true for f when f is a random permutation S-box).

- H provides two oracles, where the first oracle is still the ideal building block f of length n, but the second oracle is a random independent wn-bit permutation P. Denote such $H = (f, P)$. Note, both forward and backward queries to P are allowed (and the same is true for f when f is a random permutation S-box).

Assume further that no computationally unbounded distinguisher D making at most q queries to either F or H (for simplicity we do not split q into the number of primitive queries to f and construction queries to either E or P) can distinguish F from H with advantage greater than $\varepsilon = \varepsilon(q)$. Let us denote this by $\Delta_q(F, H) \leqslant \varepsilon$.

Now, let $F_1, \ldots, F_c$ be c independent copies of F, and $H_1, \ldots, H_c$ be c independent copies of H. Let C be the construction such that, for $L_1, \ldots, L_c$ being each either F_i or H_i, $C(L_1, \ldots, L_c)$ implements c+1 oracles, as follows. If we let $L_i = (f_i, Q_i)$ (where Q_i is either a random permutation P_i or E_i), then

$$C(L_1, \ldots, L_c) = (f_1, ..., f_c, Q_1 \circ Q_2 \circ ... \circ Q_c)$$

where $\circ$ is the composition of permutations. Namely, C is the c-time cascade of the c block ciphers E_i or random permutations P_i, which also provides oracle access to the c independent building blocks $f_1, \ldots, f_c$. Let us also denote the c-cascade of our c block ciphers by $E' = E_1 \circ \cdots \circ E_c$, and the c-cascade of random permutations P_i by $P' = P_1 \circ \cdots \circ P_c$, which by itself is just another random permutation.

It is easy to see that this construction C has a property that is called *neutralizing* by [MPR07]: whenever at least one of the H_i's is such that $L_i = H_i$ (the ideal system), meaning that Q_i is a fresh random permutation P_i, then

$$C(L_1, .., L_c) = (f_1, ..., f_c, P') = C(H_1, ..., H_c),$$

because the composition becomes random if at least one of the permutations is random. Under

such conditions, the amplification result proven in [MPR07] states that

$$\begin{aligned}\Delta_q(C(F_1,\ldots,F_c),C(H_1,...,H_c)) &= \Delta_q((f_1,....,f_c,E'),(f_1,....,f_c,P')) \\ &\leqslant 2^{c-1}\epsilon^c < (2\varepsilon)^c \end{aligned} \tag{7.3}$$

We can now apply Equation (7.3) to the 3-round linear SPN construction, where the building block f is an n-bit random permutation, and the security value $\varepsilon(q) = O(q^2/2^n)$ is established by Lemma 7.10. We then get that the resulting $3c$-round SPN construction uses c independent S-boxes $f_1 \ldots f_c$ (one per each 3 rounds) and c independent wn-bit keys $K_1 \ldots K_c$, and achieves $(q, \varepsilon'_c(q))$-security against q queries (to either the construction of the S-boxes), where $\varepsilon'_c(q) = O((q^2/2^n)^c)$.

In fact, to reach the same conclusion with a shorter key length, we could use Lemma 7.11 in place of Lemma 7.10. In this case, we get the final key of length only $cn \ll cwn$, so we save the domain expansion factor w. Thus, although we still need c independent S-boxes, for now, this version and could be viewed as a relatively advanced form of key scheduling, with very strong provable security guarantees.

Removing the dependence on q in ϵ. As with the case of PRGs, we cannot use these results as is, and need to do some manipulation of the bounds to move the dependence on the number of queries q from ϵ on q to the size of the S-box f. Let $n \geqslant 20(\log q+1)$ (or, equivalently, $2q^2 \leqslant 2^{n/10}$). Then $2\epsilon(n) = 2q^2/2^n = 2^{-0.9n}$, and hence $\varepsilon'_c(q) \leqslant (2\epsilon(n))^c = 2^{-0.9nc} = N^{-0.9c}$.

Finally, we will now no longer assume that the attacker $\mathcal{A}$ is computationally unbounded, but instead upper bound its running time by some parameter $T \geqslant q$, and say that our SPN cipher is (T, ε)-secure if no such attacker can break it with an advantage more than ϵ. With this change, we get the following restatement on our bound above.

Theorem 7.12. *If $n \geq 20(\log T+1)$, then the $3c$-round SPN construction using c independent S-boxes and c independent (either wn-bit or n-bit, depending on variant) round keys is $(T, N^{-0.9c})$-secure.*

Conjectured Hardness Amplification with Correlated Keys. Unfortunately, the hardness amplification result of [MPR07] crucially relies on the complete independence of the c S-boxes $f_1, \ldots, f_c$ and c independent round keys. In particular, unlike the much simpler PRG setting, where we managed to analyze the whole PRG construction in one go, for the case of SPNs, we currently cannot prove such strong results when the S-boxes are shared across the cascade, or keys are more correlated. The best provable result in this setting is the "computational hardness amplification" of Tessaro [Tes11], but that comes with huge degradation in the number of oracle queries q allowed by the "cascade distinguisher", leading to concrete bounds which are not useful.

In general, though, we would like to apply an appropriate hardness amplification step in practical settings, where different cascading ciphers use correlated rather than independent keys (via a key schedule used in most actual designs), or when correlated or even identical building blocks f (e.g., S-boxes) are used in different cascaded ciphers. For such pragmatic settings, we do not have any provable results such as [MPR07], and hence we state the hardness amplification step as a "conjecture" rather than "theorem" below. In particular, the concrete choice of cascading (not spelled out in the statement) is part of the conjecture. For simplicity, we also choose the final security level we desire to be 2^{-wn}, which is definitely enough for practical use, but the statement easily extends to any security level below 2^{-n}.

Conjecture 7.13 (Hardness Amplification; Informal)**.** *Let T be the desired attacker time bound, and assume that r-rounds block cipher E of length wn utilizing idealized block f of size n is $(T, 2^{-\alpha n})$-secure,* as long as $n > a \log T$ *(for some constants $a > 1$ and $\alpha < 1$). Then,* provided $n > a \log T$*, cascading E for $c = O(w/\alpha)$ times will result in a $r' = O(wr/\alpha)$-round block cipher E' which is $(T, O(T/2^{\ell(n)} + 2^{-wn}))$-secure, where $\ell(n)$ is the key length of E' under to corresponding cascading step (equal to c times the key length of E when independent keys are used).*

Ignoring the cost of the brute-force key search (against uniform attackers, for simplicity) $T/2^{\ell(n)}$ (which is expected to be negligible for our choice of parameters), the hardness amplification conjecture states that using a building block f of size n would yield *better-than-exponential-*

in-n security 2^{-wn} for sufficiently many more (still constant, assuming expansion $w = O(1)$) rounds, provided the box size n is sufficiently large.

Big-to-Small Conjecture. But now it seems natural to assume/conjecture that such a final result not only holds for "big" n but *might even be true for "small" n*! Namely, back to the original *small-box f*, we can reasonably conjecture security 2^{-wn} (plus brute-force search) for a sufficiently large constant number of rounds $r' = O(rw)$ *without assuming that this is only true when n is large.* Namely, the amplified security level 2^{-wn} is so good even if n is small, that we optimistically hope that it holds even in the small-box world, even though the supporting hardness amplification argument is no longer valid.

As discussed in Section 7.2.2, we will propose one of the strongest variants of such a conjecture. The motivation behind such a strong variant is that it gives us great security in case it happens to be true for practically used ciphers. As before, the conjecture will give a meaningful result for our purposes as long as one can decrease the size n of the 'small-box" below the threshold of $\log T$, for T independent of n. The constant $n_0 = n_0(a)$ below could be really small (e.g., $n_0 = 8$ in the case of AES), and is part of the conjecture. We also notice that we are *not* making this conjecture for all (even potentially artificial) block ciphers E', but only for specific E' resulting from applying the hardness amplification step to the basic block cipher E (for which we get our provably secure results).

Conjecture 7.14 (Big-to-Small Conjecture; Informal)**.** *Assume a block cipher E' with key length $\ell(n)$ is $(T, \varepsilon'(n))$-secure, where $\varepsilon'(n) > T/2^{\ell(n)}$, when using ideal building component of length $n \geqslant a \log T$ (for some $a > 1$). Then, for some constant $n_0 = n_0(a)$, the "scaled down" version of E' using building block f of size $n \geqslant n_0$ is still $(T, O(\varepsilon'(n))$-secure.*

We discuss this very strong conjecture below but notice that Conjectures 7.13 and 7.14 immediately imply the statement of Theorem 1.1 from the Introduction.

How Reasonable is "Big-to-Small" Conjecture? At first, this conjecture seems like a com-

plete "cheat", as we simply assume that the conclusions attained by some security arguments crucially relying on the big-box assumption $n \gg \log T$, might still hold in the small-box world when n is a constant. But let us observe a couple of things. First, we already mentioned that we do not need such a strong conjecture: many weaker conjectures will yield meaningful variants of Theorem 1.1, provided they allow one to decrease the size n of the "small-box" below the threshold value $\log T$. Second, since the construction of E' is the same for all n, it is natural that its security smoothly changes with n, without any huge jumps at certain levels, as long as the exhaustive key search is infeasible (this is why we assumed $\varepsilon'(n) > T/2^{\ell(n)}$). In particular, under this reasonable assumption, we certainly allow the assumed success probability $\varepsilon'(n)$ to grow as the box f becomes smaller. So the only really big assumption is the fact that we kept the running time of the attacker at the same level T, even though when T becomes larger than 2^n, the attacker can suddenly evaluate our ideal component f (e.g., S-box) on *all* 2^n inputs. Third, given our current inability to built unconditionally block ciphers from only small components, it seems that some kind of "big-to-small" conjecture must be required, but we tried to make it as crisp and clean as we could, while additionally proving as many things around it as possible with the existing techniques. And, finally, the kinds of constructions we get when applying this conjecture to the SPN ciphers are exactly the SPN ciphers used in practice, and *believed to be secure*. So one can use this conjecture as a clean and formal way to isolate exactly the kind of "leap of faith" we are making in the real world in assuming these ciphers are secure.

Aside from these reasonable, but still rather limited, justifications at this stage we don't have any other theoretical justification for this strong "Big-to-Small Conjecture", and view this as an exciting direction for future research. In particular, given that coupling this strong conjecture with (rather mild and believable) hardness amplification step gives us the amazing conclusion of Theorem 1.1, which in turn implies plausible security for many SPN-based ciphers, we believe studying this new and non-standard conjecture is extremely reasonable and well-motivated.

7.3.4 Putting the Pieces Together

As mentioned earlier, Dodis *et al.*[CDK+18] proved results that addressed the problem of "domain extension" of block ciphers. In particular, they showed that a 3-round SPN is $(T, 2^{-\alpha n})$-secure when $n > 2\log T/(1-\alpha)$ (so that $T^2/2^n \leqslant 2^{-\alpha n}$). Thus, cascading it c times gives us $3c$-round SPN with conjectured $(T, T/2^{\ell(n)} + 2^{-\Omega(cn)})$-security, where $\ell(n)$ is our final key length, and this is true even for small values of n (governed by constant n_0 which is part of the conjecture). To get this close to the practical SPN designs, let us write $T = 2^t$, and assume we use correlated key schedule with final key length $\ell(n) = wn$, and, for simplicity, ideal hardness amplification is true even with best possible $\alpha \approx 1$. Then we get (very ambitious) conjectured $(2^t, 2^{t-wn} + 2^{-cn})$-security in $3c$ rounds. In particular, optimistically setting $n = 8$ and $wn = 128$ for the case of AES, we could get ambitious $(2^t, 2^{t-128} + 2^{-8c})$-security in $3c$ rounds. Assume $c \leqslant 8$ and $t = 64$ is good enough for practical use, we simplify this to an amazingly simple, but powerful, conclusion of our small-box cryptography framework:

$$\boxed{\textit{3c-round variant of 128-bit AES with 8-bit S-boxes which is } (2^{64}, 2^{-8c})\textit{-secure}}$$

In particular, setting $c = 10/3$, would already yield respectable one-in-hundred-million security in 10 rounds (the number of real AES rounds), while setting $c = 8$ would give excellent 2^{-64} security in 24 rounds.

While the above "back-of-the-envelope" calculations were a bit ad hoc and likely quite optimistic, they demonstrate several very attractive features of our framework, especially in comparison to its "big-box" counterpart. First, such calculations *can* be easily made (although more research is needed in estimating or conjecturing the right constants hidden/underspecified in Theorem 1.1). Second, such calculations give meaningful conjectured security of *actually used* ciphers. Third, for the first time, we see that our conjectured bounds — even when ambitiously good — were on the *pessimistic* side, predicting either more rounds or a lower level of conjectured security than what is believed in practice. This is exactly what we expect from a sound theory,

as we don’t want such a theory to make predictions contradicted by reality.

Part III

Searchable Encryption

8 | Searchable Encryption

This chapter is based on joint work with Erik Aronesty, David Cash, Yevgeniy Dodis, Daniel H. Gallancy, Christopher Higley, and Oren Tysor that appeared in PKC 2022 [ACD+22a]. Passages are taken verbatim from the full version of this paper [ACD+22b]. This chapter considers the problem of searchable encryption, in the public-key setting.

8.1 Preliminaries

Before we look at the construction, we will briefly state the necessary mathematical background. In Section 8.1.1, we will define bilinear maps and bilinear groups. In Section 8.1.2 we will look at the various hardness assumptions that we use in this work.

8.1.1 Bilinear groups.

We use bilinear maps in our constructions.[1] We briefly review their properties in this section. Interested readers can refer [DBS04] for a more comprehensive treatment.

Consider two (multiplicative) cyclic groups G and G_1 of prime order p. Let g be a generator of G. Roughly speaking, a mapping is **bilinear** if it is linear with respect to each of its variables:

Definition 8.1. An (admissible) bilinear map is a map $e : G \times G \mapsto G_1$ with the following properties:

[1] For simplicity of exposition, we will use symmetric bilinear maps. However, all our assumptions and constructions easily extend to the asymmetric variant; see [BF03].

1. **Bilinear:** for all $u, v \in G$ and $x, y \in \mathbb{Z}$, we have $e(u^x, v^y) = e(u, v)^{xy}$.

2. **Non-degenerate:** $e(g, g) \neq 1$.

3. **Computable:** there is an efficient algorithm to compute $e(u, v)$ for all $u, v \in G$.

We say that a group G is bilinear if the group action in G is efficiently computable and there exists a group G_1 and an admissible bilinear map $e : G \times G \mapsto G_1$. Henceforth, we shall use G^* to stand for $G \backslash \{1_G\}$. Such maps can be constructed from Weil and Tate pairings on elliptic curves or abelian varieties [BF03, JN01, Gal01]. DDH is easy in Bilenear groups. Given two group elements $A = g^a, B = g^b \in G$, we define the $\mathsf{DH}(A, B) = \mathsf{DH}(g^a, g^b) = g^{ab} \in G$. Note that an admissible bilinear map provides an algorithm for solving the decisional Diffie-Hellman problem (DDH) in G. Specifically, to determine whether (g, A, B, R) is a *DDH tuple*, meaning check if $R = \mathsf{DH}(A, B)$. Indeed, $R = \mathsf{DH}(A, B)$ iff $e(A, B) = e(g, R)$, which is efficiently checkable in bilinear groups.

8.1.2 Complexity assumptions

We now state the hardness assumptions on which our constructions are based. In what follows, we let G be a bilinear group of prime order p, and let g be its generator.

8.1.2.1 Bilinear Diffie-Hellman Assumption

Our basic and unidirectional EVRF constructions will rely on the Bilinear Decisional Diffie-Hellman (BDDH) assumption from the original Boneh-Franklin paper [BF03]. The BDDH problem on G asks: given $\left(g, g^a, g^b, g^r\right) \in (G^*)^4$ as input, distinguish the value $e(g, g)^{abr}$ from random element $h \leftarrow\!\!\$\, G_1$. Formally, an algorithm $\mathcal{B}$ has advantage ϵ in solving BDDH in G if

$$\left|\Pr\left[\mathcal{B}(g, g^a, g^b, g^r, e(g, g)^{abr}) = 1\right] - \Pr\left[\mathcal{B}(g, g^a, g^b, g^r, g_1) = 1\right]\right| \leq \epsilon,$$

where the probability is over the internal coin tosses of $\mathcal{A}$ and the choice of $a, b, r \in \mathbb{Z}_p^*$ and $g_1 \in G_1$.

Definition 8.2. (BDDH assumption) We say that BDDH *assumption* holds in $\mathbb{G}$ if every PPT algorithm $\mathcal{B}$ has advantage at most $\epsilon(k) = \text{negl}(k)$ in solving the BDDH problem in G.

8.1.2.2 Extended BDDH assumption

Our one-time bidirectional delegatable EVRF construction will rely on the *extended* Bilinear Decisional Diffie-Hellman (eBDDH) that we introduce, where the attacker is additionally given the value $g^{1/a} \in G^*$. Formally, an algorithm $\mathcal{B}$ has advantage ϵ in solving eBDDH in G if

$$\left|\Pr\left[\mathcal{B}(g, g^{1/a}, g^a, g^b, g^r, e(g,g)^{abr}) = 1\right] - \Pr\left[\mathcal{B}(g, g^{1/a}, g^a, g^b, g^r, g_1) = 1\right]\right| \leq \epsilon,$$

where the probability is over the internal coin tosses of $\mathcal{A}$ and the choice of $a, b, r \in \mathbb{Z}_p^*$ and $g_1 \in G_1$.

Definition 8.3. (eBDDH assumption) We say that *extended* eBDDH *assumption* holds in $\mathbb{G}$ if every PPT algorithm $\mathcal{B}$ has advantage at most $\epsilon(k) = \text{negl}(k)$ in solving the eBDDH problem in G.

8.1.2.3 Inversion-Oracle BDDH assumption

Our multi-time bidirectional delegatable EVRF construction will rely on the *inversion-oracle* Bilinear Decisional Diffie-Hellman (iBDDH) that we introduce, where the attacker is additionally given the oracle $O_a(h) = h^{1/a} \in G^*$. Formally, an algorithm $\mathcal{B}$ has advantage ϵ in solving iBDDH in G if

$$\left|\Pr\left[\mathcal{B}^{O_a(\cdot)}(g, g^a, g^b, g^r, e(g,g)^{abr}) = 1\right] - \Pr\left[\mathcal{B}^{O_a(\cdot)}(g, g^a, g^b, g^r, g_1) = 1\right]\right| \leq \epsilon,$$

where the probability is over the internal coin tosses of $\mathcal{A}$ and the choice of $a, b, r \in \mathbb{Z}_p^*$ and $g_1 \in G_1$.

Definition 8.4. (iBDDH assumption) We say that *inversion-oracle* BDDH *assumption* (iBDDH) holds in $\mathbb{G}$ if every PPT algorithm $\mathcal{B}$ has advantage at most $\epsilon(k) = \text{negl}(k)$ in solving the iBDDH problem in G.

Clearly, iBDDH assumption is stronger than eBDDH (as $g^{1/a} = \mathcal{O}_a(g)$), which in turn is stronger than the well believed BDDH assumption.

$$\text{iBDDH} \implies \text{eBDDH} \implies \text{BDDH}$$

These assumptions can be proven secure in the Generic Group Model. We now present an analysis of the iBDDH assumption in the generic group model.

Now, we examine the iBDDH assumption in the generic group model [Sho97]. In the generic group model, elements of G, G_1 are encoded as unique random strings. We define an injective function $\theta : \mathbb{Z}_p \mapsto \{0,1\}^*$ which maps $a \in \mathbb{Z}_p$ to the string encoding of the group element $g^a \in G$. Similarly, we define $\theta_1 : \mathbb{Z}_p \mapsto \{0,1\}^*$ for G_1. The encodings are such that non-group operations are meaningless. Typically, there exist three oracles - one which computes the group action in G, another which computes the group action in G_1, and a third that compute the bilinear pairing $e : G \times G \mapsto G_1$. In our case, there exists a fourth oracle that models the inversion oracle $\mathcal{O}_a(\cdot)$.

Theorem 8.5. *Let $\mathcal{A}$ be an algorithm that solves the* iBDDH *problem. Assume that $a \in \mathbb{Z}_p^*$, $b, c, r \in \mathbb{Z}_p$, and the encoding functions θ, θ_1 are chosen at random. If $\mathcal{A}$ makes at most q_G queries to the four*

oracles, then

$$\left|\Pr\left[\mathcal{A}^{O_a(\cdot)}\begin{pmatrix} p, \theta(1), \\ \theta(a), \theta(b), \theta(c), \\ \theta_1(\Gamma_0), \theta_1(\Gamma_1) \end{pmatrix} = \beta \,\middle|\, \begin{matrix} \beta \leftarrow \{0,1\}; \\ a \leftarrow \mathbb{Z}_p^*, b, c, r \leftarrow \mathbb{Z}_p, \\ \Gamma_\beta = abc, \Gamma_{1-\beta} = r \end{matrix}\right] - \frac{1}{2}\right| \leq \frac{2 \cdot (q_G + 6)^2 (q_G - 3)}{p}$$

Proof. Instead of letting $\mathcal{A}$ interact with the actual oracles, we play the following game.

We begin by maintaining two dynamic lists L, L'. Informally, L contains the information that $\mathcal{A}$ has garnered about group elements in G, while L' contains information about those elements in G_1. Formally, we have

$$L = \{(F_i, s_i) : i = 0, \ldots, t-1\},$$
$$L_1 = \{(F_i', s_i') : i = 0, \ldots, t'-1\}.$$

Here $s_i, s_i' \in \{0,1\}^*$ are random encodings and
$F_i, F_i' \in \mathbb{Z}_p[A, B, C, A^{-1}, \Gamma_0, \Gamma_1]$ are multivariate *Laurent* polynomials in $A, B, C, A^{-1}, \Gamma_0, \Gamma_1$. Note that this is a departure from typical modeling which models F_i as polynomials with only positive degree elements. However, we need Laurent polynomials here to capture the inversion oracle.

In the beginning of the game, the lists are initialized as follows: $F_0 = 1, F_1 = A, F_2 = B, F_3 = C$, $F_0' = \Gamma_0, F_1' = \Gamma_1$. The corresponding encodings are set to arbitrary distinct strings in $\{0,1\}^*$. At this state, the lists have sizes $t = 4, t' = 2$. The total length of lists at a step $\tau \leq q_G$ in the game must satisfy:

$$t + t' = \tau + 6, \tag{8.1}$$

and it is easy to note that this condition is met at the start of the game.

We start the game by providing $\mathcal{A}$ with encodings $s_0, s_1, s_2, s_3, s_0', s_1'$. Algorithm $\mathcal{A}$ begins to

make oracle queries and we respond as follows:

- Group Actions: $\mathcal{A}$ provides a multiply/divide bit and two operands $0 \leq i, j < t$ corresponding to two encodings s_i, s_j. We first compute $F_t = F_i \pm F_j$ depending on the choice of the bit. We then check to see if $\exists \ell < t$ such that $F_\ell = F_t$. If yes, we set $s_t = s_\ell$. Otherwise, we set s_t to a random string in $\{0,1\}^* \setminus \{s_0, \ldots, s_{t-1}\}$. Then increment t by 1 and return s_t to $\mathcal{A}$. Similarly, we handle operations in G_1, except that the operation is on the other list L'.

- Bilinear Pairing: $\mathcal{A}$ provides $0 \leq i, j < t'$ indicating operands s_i, s_j. First compute $F'_{t'} = F_i \cdot F_j$. We then check to see if $\exists \ell < t'$ such that $F'_\ell = F'_{t'}$. If yes, we set $s'_{t'} = s'_\ell$. Otherwise, we set $s'_{t'}$ to a random string in $\{0,1\}^* \setminus \{s'_0, \ldots, s'_{t'-1}\}$. Then increment t' by 1 and return $s'_{t'}$ to $\mathcal{A}$.

- Inversion Oracle: $\mathcal{A}$ provides $0 \leq i < t$ indicating operand s_i. First compute $F_t = F_i/A$. We then check to see if $\exists \ell < t$ such that $F_\ell = F_t$. If yes, we set $s_t = s_\ell$. Otherwise, we set s_t to a random string in $\{0,1\}^* \setminus \{s_0, \ldots, s_{t-1}\}$. Then increment t by 1 and return s_t to $\mathcal{A}$.

After making at most q_G queries, $\mathcal{A}$ halts with a guess $\hat{\beta} \in \{0,1\}$. Then we choose $a \leftarrow \mathbb{Z}_p^*, b, c, r \leftarrow \mathbb{Z}_p$. Then, consider $\Gamma_\beta \leftarrow abc$, $\Gamma_{1-\beta} \leftarrow r$ for both choices of β. By sampling the values only after $\mathcal{A}$'s guess bit, the simulation does not reveal anything about β, unless our indeterminates give rise to some non-trivial equality relation for the sampled values. Specifically, $\mathcal{A}$ wins the game if for any $F_i \neq F_j$ or any $F'_i \neq F'_j$, one of the following conditions hold:

1. $F_i(a, b, c, a^{-1}, abc, r) - F_j(a, b, c, a^{-1}, abc, r) = 0$
2. $F_i(a, b, c, a^{-1}, r, abc) - F_j(a, b, c, a^{-1}, r, abc) = 0$
3. $F'_i(a, b, c, a^{-1}, abc, r) - F'_j(a, b, c, a^{-1}, abc, r) = 0$
4. $F'_i(a, b, c, a^{-1}, r, abc) - F'_j(a, b, c, a^{-1}, r, abc) = 0$

Let us look at the degree of the polynomials. For all i, $\deg(F_i) \leq t-3$, and $\deg(F'_i) \leq 2(t-3)$. In other words, each F_i is a polynomial in A, B, C, A^{-1}, ABC, R whose degree is at most $t-3$. Consider the polynomial: $f_i = F_i \cdot A^{t-4}$. Note that f_i is a polynomial only in A, B, C, ABC, R and $\deg(f') \leq t-3$. Further, every root of F_i is a root of f_i and there are at most $t-3$ roots of f_i. Similarly, consider the polynomial $f'_i = F'_i \cdot A^{2t-8}$. The degree of f'_i is at most $2t-6$ and every root of F'_i is a root of f'_i, and there are at most $2t-6$ roots of f'_i. We can therefore apply Schwartz-Zippel Lemma [Sch80] to get that for all i, j, $\Pr[[]F_i - F_j = 0] \leq \Pr[[]f_i - f_j = 0] \leq (t-3)/p, \Pr[[]F'_i - F'_j] \leq \Pr[[]f'_i - f'_j = 0] \leq 2(t-3)/p$. Therefore, $\mathcal{A}$'s advantage is:

$$\begin{aligned}
\varepsilon &\leq 2 \cdot \left(\binom{t}{2} \frac{t-3}{p} + \binom{t'}{2} \frac{2 \cdot (t-3)}{p} \right) \\
&< (t+t')^2 \cdot \frac{2 \cdot (t-3)}{p} \\
&< (q_G + 6)^2 \cdot \frac{2 \cdot (t-3)}{p} \\
&< (q_G + 6)^2 \cdot \frac{2 \cdot (q_G - 3)}{p} = O\left(\frac{q_G^3}{p} \right)
\end{aligned}$$

□

8.2 Encapsulated Search Index

We begin by formally introducing the new primitive of *standard* Encapsulated Search Index in Section 8.2.1, defining its syntax and security. We then present extensions to this primitive, adding features such as distribution (section 8.2.3), delegatability (section 8.2.4), and updatability (section 8.2.5).

8.2.1 Standard Encapsulated Search Index

For visual simplicity, for the remainder of this section we will use upper-case letters (D, E, Y, etc.) to denote objects whose size can depend on the size of document D (with the exception of various keys SK, PK, etc.), and by lower-case letters (c, s, r, z, w, etc.) objects whose size is constant.

In the definition below, we let k be a security parameter, PPT stand for probabilistic polynomial-time Turing machines, $\text{poly}(k)$ to denote a polynomial function, and $\text{negl}(\kappa)$ to refer to a negligible function in the security parameter k.

Definition 8.6. An Encapsulated Search Index (ESI) is a tuple of PPT algorithms $\text{ESI} = (\text{KGen}, \text{Prep}, \text{Index}, \text{S-Split}, \text{S-Core}, \text{Finalize})$ such that:

- $(PK, SK) \leftarrow\!\!\$\ \text{KGen}(1^\kappa)$: outputs the public/secret key pair.
- $(s, c) \leftarrow\!\!\$\ \text{Prep}(PK)$: outputs compact representation c, and trapdoor s.
- $\text{Index}(s, D) = E$: outputs the encrypted index E for a document D using the trapdoor s.
- $\text{S-Split}(PK, c') = r'$: outputs a handle r' from the representation c'.
- $\text{S-Core}(SK, r', w) = z'$: outputs a partial result z' from the handle r'.
- $\text{Finalize}(PK, E', c', z', w) = \beta \in \{0, 1, \perp\}$: outputs 1 if the word w is present in the original document D, 0 if not present, and $\perp$ if the partial output z' is inconsistent.

Before we define the security properties, it is useful to define the following shorthand functions:

- $\text{BldIdx}(PK, D) = (\text{Index}(s, D), c)$, where $(s, c) \leftarrow\!\!\$\ \text{Prep}(PK)$.
- $\text{S-Prove}(PK, SK, c, w) = \text{S-Core}(SK, \text{S-Split}(PK, c), w)$.
- $\text{Search}(PK, SK, (E, c), w) = \text{Finalize}(PK, E, c, \text{S-Prove}(SK, c, w), w)$.

We require the following security properties from this primitive:

1. **Correctness**: with prob. 1 (resp. $(1 - \mathrm{negl}(\kappa))$) over randomness of KGen and Prep, for all documents D and keywords $w \in D$ (resp. $w \notin D$):

$$\textsc{Search}(PK, SK, \textsc{BldIdx}(PK, D), w) = \begin{cases} 1 & \text{if } w \in D \\ 0 & \text{if } w \notin D \end{cases}$$

2. **Uniqueness**: there exist no values (PK, E, c, z_1, z_2, w) such that $b_1 \neq \perp$, $b_2 \neq \perp$ and $b_1 \neq b_2$, where:

$$b_1 = \textsc{Finalize}(PK, E, c, z_1, w); \quad b_2 = \textsc{Finalize}(PK, E, c, z_2, w)$$

3. **CCA Security**: We require that for any PPT algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ the following holds, where $\mathcal{A}$ does not make the query S-$\textsc{Prove}(PK, SK, c^*, w)$ with $w \in (D_1 \setminus D_2) \cup (D_2 \setminus D_1)$ and $|D_1| = |D_2|$, for variables SK, c^*, D_1, D_2, w defined below:

$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} (PK, SK) \leftarrow\!\!\$\ \textsc{KGen}(1^\kappa); \\ (D_1, D_2, st) \leftarrow\!\!\$\ \mathcal{A}_1^{\text{S-}\textsc{Prove}(PK,SK,\cdot,\cdot)}(PK); \\ b \leftarrow\!\!\$\ \{0,1\}; \\ (E^*, c^*) \leftarrow\!\!\$\ \textsc{BldIdx}(PK, D_b); \\ b' \leftarrow\!\!\$\ \mathcal{A}_2^{\text{S-}\textsc{Prove}(PK,SK,\cdot,\cdot)}(E^*, c^*, st) \end{array} \right] \leq \frac{1}{2} + \mathrm{negl}(\kappa)$$

4. **Privacy-Preserving** [2]: We require that for any PPT Algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ which outputs documents D_1, D_2 such that $|D_1| = |D_2|$ for variables D_1, D_2 defined below, the follow-

[2] It is easy to see that our syntax guarantees that any ESI construction is *unconditionally* **Privacy-Preserving** (even with knowledge of SK), for the simple reason that Prep that produces c does not depend on the input document D. Thus, we will never explicitly address this property, but list it for completeness, as it is important for our motivating application.

ing holds:

$$\Pr\left[\begin{array}{c|c} b = b' & \begin{array}{c} (PK, SK) \leftarrow\!\!\$\ \textsc{KGen}(1^\kappa); \\ (D_1, D_2, st) \leftarrow\!\!\$\ \mathcal{A}_1(PK, SK); \\ b \leftarrow\!\!\$\ \{0,1\}; \\ (E^*, c^*) \leftarrow\!\!\$\ \textsc{BldIdx}(PK, D_b); \\ b' \leftarrow\!\!\$\ \mathcal{A}_2(c^*, st) \end{array} \end{array}\right] \leq \frac{1}{2} + \text{negl}(\kappa)$$

Remark 4. We want to ensure that an honest representation c_1 will not collide with another honest representation c_2. With this, we can ensure that honestly generated documents do not conflict. If there is a non-trivial chance of such a collision, then one can simply generate c_2 until collision with the challenge c_1. With this collision, and with knowledge of trapdoor T_2, one can trivially break security.

Remark 5. For efficiency, we will want Search to run in time $O(\log N)$ or less, where N is the size of the document D. In fact, our main construction will have S-Prove run in time $O(1)$, independent of the size of the document, and Finalize would run in time at most $O(\log N)$, depending on the non-cryptographic data structure we use.

8.2.2 Extensions to ESI

Threshold ESI. We extend the definition of the standard Encapsulated Search Index to achieve support for distributed token generation. To do this, we introduce a new algorithm called KG-Verify that aims to verify if the output of the KGen algorithm is correct, and replace Finalize with two more fined-grained procedures S-Verify and S-Combine. The formal discussion about the syntax and security of this primitive can be found in Section 8.2.3.

Delegatable ESI. We can also extend the definition of the standard Encapsulated Search Index to achieve support for delegation. Informally, Encapsulated Search Index is delegatable if there

are two polynomial-time procedures S-Del, S-Check that work as follows: S-Del that achieves the delegation wherein it takes as input a representation c corresponding to one key pair and produces a representation c' corresponding to another key pair; S-Check helps verify if a delegation was performed correctly. The formal discussion about the syntax and security of this primitive, including several definitional subtleties, can be found in Section 8.2.4.

Updatable ESI. We can further extend the definition of the standard Encapsulated Search Index to support a use-case where one might want to remove a word, or add a word to the document D, without having to necessarily recompute the entire index. To achieve this, we need an additional algorithm called Update that can produce a new index E' after performing an `action` relating to word w in original index E, using the same token z_w used for searching. The formal discussion about the syntax and security of this primitive can be found in Section 8.2.5.

8.2.3 Threshold Encapsulated Search Index

Definition 8.7. A (t, n)-threshold Encapsulated Search Index (TESI) is a tuple of PPT algorithms TESI = (KGen, KG-Verify, Prep, Index, S-Split, S-Core, S-Verify, S-Combine) such that:

- $(PK, \mathbf{SK} = (sk_1, \ldots, sk_n), \mathbf{VK} = (vk_1, \ldots, vk_n)) \leftarrow\!\!\$\ \text{KGen}(1^\kappa, t, n)$: outputs the public key PK, a vector of secret shares $\mathbf{SK}$, and public key PK.
- KG-Verify$(PK, \mathbf{VK}) = \beta \in \{0, 1\}$: verifies that the output of KGen is indeed valid.
- $(s, c) \leftarrow\!\!\$\ \text{Prep}(PK)$: outputs compact representation e, and trapdoor s.
- Index$(s, D) = E$: outputs the encrypted index E for a document D using the trapdoor T.
- S-Split$(PK, n, c') = (r'_1, \ldots, r'_n)$: outputs n handles $r'_1, \ldots, r'_n$ from c'.
- S-Core$(sk_i, r'_i, w) = z'_i$: outputs partial result on input x using handle r'_i and secret key share sk_i.

- S-$\textsc{Verify}(PK, vk_i, z'_i, w) = \beta \in \{0,1\}$: verifies that the share z'_i produced by party i is valid.
- S-$\textsc{Combine}(PK, E', c', z'_{i_1}, \ldots, z'_{i_t}, w) = \beta \in \{0,1\}$: uses the partial shares to determine if the word w is present in D or not [3].

Before we define the security properties, it is useful to define the following shorthand functions:

- $\textsc{BldIdx}(PK, D) = (\textsc{Index}(s, D), c)$ where $(s, c) \leftarrow\!\!\$\ \textsc{Prep}(PK)$.
- S-$\textsc{Prove}(\mathbf{SK}, i, c, w) =$ S-$\textsc{Core}(sk_i, r_i, w)$ where $r_1, \ldots, r_n =$ S-$\textsc{Split}(PK, n, c)$
- $\textsc{Search}(\mathbf{SK}, i_1, \ldots, i_t, (E, c), w)$: For $j = 1, \ldots, t$ let $z_{i_j} =$ S-$\textsc{Prove}(\mathbf{SK}, i, c, w)$.
  Output $\perp$ if, for some $1 \le j \le t$, S-$\textsc{Verify}(PK, vk_{i_j}, z_{i_j}, w) = 0$.
  Otherwise, output S-$\textsc{Combine}(PK, E, c, z_{i_1}, \ldots, z_{i_t}, w)$.

We require the following security properties from this primitive:

1. **Correctness**:
   (a) with prob. 1 over randomness of $(PK, \mathbf{SK}, \mathbf{VK}) \leftarrow\!\!\$\ \textsc{KGen}(1^\kappa, t, n)$,
   KG-$\textsc{Verify}(PK, \mathbf{VK}) = 1$.
   (b) with prob. 1 (resp. $(1 - \text{negl}(\kappa))$) over randomness of KGen and Prep, for all documents D and keywords $w \in D$ (resp. $w \notin D$):

$$\textsc{Search}(\mathbf{SK}, i_1, \ldots, i_t, \textsc{BldIdx}(PK, D), w) = \begin{cases} 1 & w \in \text{ if } D \\ 0 & w \text{ if } \notin D \end{cases}$$

2. **Uniqueness**: there exist no values $(PK, \mathbf{VK}, E, c, Z_1, Z_2, w)$ where
   $Z_1 = ((i_1, z_{i_1}), \ldots, (i_t, z_{i_t}))$ and $Z_2 = ((j_1, z_{j_1}), \ldots, (j_t, z_{j_t}))$. st

[3] Without loss of generality, we will always assume that all the t partial evaluations z_i satisfy S-$\textsc{Verify}(PK, vk_i, z'_i) = 1$ (else, we output $\perp$ before calling S-Combine).

(a) $\text{Gen-Vfy}(PK, \mathbf{VK}) = 1$,

(b) for $k = 1, \ldots, t$:

- $\text{S-Verify}(PK, vk_{i_k}, z_{i_k}, w) = 1$.
- $\text{S-Verify}(PK, vk_{j_k}, z_{j_k}, w) = 1$,.

(c) and

$$\text{S-Combine}(PK, E, c, z_{i_1}, \ldots, z_{i_t}, w) \neq \text{S-Combine}(PK, E, c, z_{j_1}, \ldots, z_{j_t}, w)$$

3. **CCA Security**: We require that for any PPT algorithm $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$ the following holds, where $\mathcal{A}$ does not make the query $\text{S-Prove}(\mathbf{SK}, j, c^*, w)$ with $w \in (D_1 \setminus D_2) \cup (D_2 \setminus D_1)$, $|D_1| = |D_2|$ and $j \notin \{i_1, \ldots, i_{t-1}\}$, for variables $\mathbf{SK}, i_1, \ldots, i_{t-1}, c^*, D_1, D_2$ defined below:

$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} \{i_1, \ldots, i_{t-1}, st\} \leftarrow\!\!\$\ \mathcal{A}_0(1^\kappa, t, n) \\ (PK, \mathbf{SK}, \mathbf{VK}) \leftarrow\!\!\$\ \text{KGen}(1^\kappa, t, n); \\ (D_1, D_2, st) \leftarrow\!\!\$\ \mathcal{A}_1^{\text{S-Prove}(\mathbf{SK}, \cdot, \cdot, \cdot)}(PK, \mathbf{VK}, \mathbf{SK}', st); \\ b \leftarrow\!\!\$\ \{0,1\}; (E^*, c^*) = \text{BldIdx}(PK, D_b); \\ b' \leftarrow\!\!\$\ \mathcal{A}_2^{\text{S-Prove}(\mathbf{SK}, \cdot, \cdot, \cdot)}(E^*, c^*, st) \end{array} \right] \leq \frac{1}{2} + \text{negl}(\kappa)$$

where $\mathbf{SK}' = (sk_{i_1}, \ldots, sk_{i_{t-1}})$.

4. **Privacy-Preserving**: We require that for any PPT Algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ which outputs documents D_1, D_2 such that $|D_1| = |D_2|$ for variables D_1, D_2 defined below, the following holds:

$$\Pr\left[\, b = b' \;\middle|\; \begin{array}{c} (PK, \mathbf{SK}, \mathbf{VK}) \leftarrow\!\!\$\ \textsc{KGen}(1^\kappa, t, n); \\ (D_1, D_2, st) \leftarrow\!\!\$\ \mathcal{A}_1(PK, \mathbf{SK}, \mathbf{VK}); \\ b \leftarrow\!\!\$\ \{0,1\}; \\ (c^*, E^*) = \textsc{BldIdx}(PK, D_b); \\ b' \leftarrow\!\!\$\ \mathcal{A}_2(c^*, st) \end{array} \right] \leq \frac{1}{2} + \mathrm{negl}(\kappa)$$

It is again easy to see that the above primitive is unconditionally **Privacy-Preserving**, even with knowledge of **SK**, since Prep that produces e does not depend on the input document D.

8.2.4 Delegatable Encapsulated Search Index

In this section, we extend the definition of the standard Encapsulated Search Index to achieve delegatability. Our definition will capture 3 security levels. Similar levels of security will also appear in the DEVRF definition (discussed in Section 8.5.1).

Definition 8.8. An Encapsulated Search Index ESI = (KGen, Prep, Index, S-Split, S-Core, Finalize) is also delegatable if there exists polynomial-time procedure S-Del, S-Check such that:

- S-Del$(SK_1, c_1, SK_2) = c_2$: outputs a representation c_2 corresponding to key pair (PK_2, SK_2) from a representation c_1 corresponding to key pairs (PK_1, SK_1).[4]
- S-Check$(PK_1, c_1, PK_2, c_2) = \beta \in \{0,1\}$:

Before we define the security properties, it is useful to define the following shorthand functions:

- BldIdx$(PK, D) =$ (Index$(s, D), c$) where $(s, c) \leftarrow\!\!\$$ Prep(PK).
- S-Prove$(SK, c, w) =$ S-Core$(SK,$ S-Split$(PK, c), w)$.
- Search$(SK, (E, c), w) =$ Finalize$(PK, E, c,$ S-Prove$(SK, c, w), w)$.

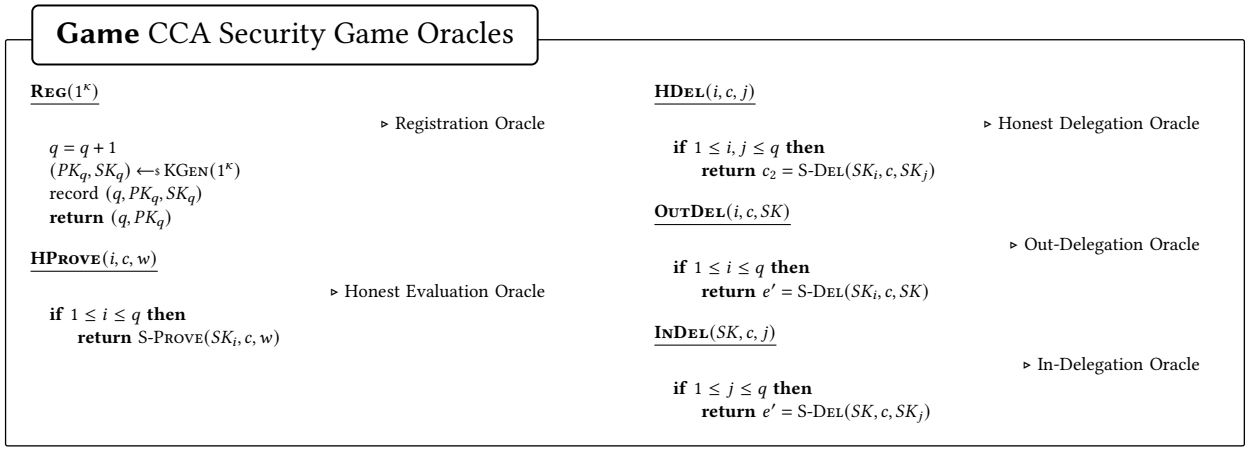

Figure 8.1: The list of oracles that an adversary $\mathcal{A}$ has access to in the CCA security game. Here c is the compact representation.

In addition to the standard Encapsulated Search Index properties of **Correctness**,**Uniqueness**, and **Privacy-Preserving**, we require the following security properties from a delegatable Encapsulated Search Index:

1. **Delegation-Completeness**: for any valid (PK_1, SK_1), (PK_2, SK_2), and compact representation c_1

$$\text{S-DEL}(SK_1, c_1, SK_2) = c_2 \implies \text{S-CHECK}(PK_1, c_1, PK_2, c_2) = 1$$

2. **Delegation-Soundness**: for any valid (PK_1, SK_1), (PK_2, SK_2), encrypted index E, and compact representations c_1, c_2

$$\text{S-CHECK}(PK_1, c_1, PK_2, c_2) = 1 \implies \forall w \ \text{SEARCH}(SK_1, E, c_1, w) = \text{SEARCH}(SK_2, E, c_2, w)$$

3. **CCA Security**: We require that for any PPT algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, with access to the oracles as defined in Figure 8.1 which outputs documents D_1, D_2 such that $|D_1| = |D_2|$ for variables D_1, D_2 defined below, and where legality of $\mathcal{A}$ and appropriate delegation oracle(s)

[4]By default, this algorithm takes as input the secret key of the other party. We can consider a publicly-delegatable algorithm that takes as input only the public key of the second party.

are defined separately, the following holds:

$$\Pr\left[\, b = b' \;\middle|\; \begin{array}{c} (1, PK_1) \leftarrow_{\$} \textsc{Reg}(1^\kappa); \\ (D_1, D_2) \leftarrow_{\$} \mathcal{A}_1^{\textsc{Reg},\textsc{HProve},\mathcal{O}}(PK_1); \\ b \leftarrow_{\$} \{0,1\}; \\ (E^*, c^*) = \textsc{BldIdx}(PK, D_b); \\ b' \leftarrow_{\$} \mathcal{A}_2^{\textsc{Reg},\textsc{HProve},\mathcal{O}}(E^*, c^*) \end{array} \right] \leq \frac{1}{2} + \mathrm{negl}(\kappa)$$

(a) **Basic CCA Security**: $\mathcal{O} = \{\textsc{HDel}\}$.

Legality of $\mathcal{A}$: no call to $\textsc{HProve}(i, c, w)$ st

$\textsc{S-Check}(PK_1, c^*, PK_i, c) = 1$ and $w \in (D_1 \setminus D_2) \cup (D_2 \setminus D_1)$.

(b) **Uni CCA Security**: $\mathcal{O} = \{\textsc{HDel}, \textsc{OutDel}\}$.

Legality of $\mathcal{A}$: no call to $\textsc{HProve}(i, c, w)$ or $\textsc{OutDel}(i, c, *)$ st

$\textsc{S-Check}(PK_1, c^*, PK_i, c) = 1$ and $w \in (D_1 \setminus D_2) \cup (D_2 \setminus D_1)$.

(c) **Bi CCA Security**: $\mathcal{O} = \{\textsc{HDel}, \textsc{OutDel}, \textsc{InDel}\}$.

Legality of $\mathcal{A}$: no call to $\textsc{HProve}(i, c, w)$ or $\textsc{OutDel}(i, c, *)$ st

$\textsc{S-Check}(PK_1, c^*, PK_i, c) = 1$ and $w \in (D_1 \setminus D_2) \cup (D_2 \setminus D_1)$.

Remark 6. It is easy to see that **Delegation-Completeness** and **Delegation-Soundness** imply **Delegation-Correctness** which is defined as follows: for any valid (PK_1, SK_1), (PK_2, SK_2), and compact representation c_1

$$\textsc{S-Del}(SK_1, c_1, SK_2) = c_2 \implies \forall w\ \textsc{Search}(SK_1, E, c_1, w) = \textsc{Search}(SK_2, E, c_2, w)$$

8.2.5 Updatable Encapsulated Search Index

Another useful extension to Encapsulated Search Index would be to support operations that help update the index. However, note that the process of updating should not reveal information about the underlying keywords. To achieve this, we extend the standard definition.

Definition 8.9. An Encapsulated Search Index ESI = (KGen, Prep, Index, S-Split, S-Core, Finalize) is also updatable if there exists polynomial-time procedures Update such that:

- $E' \cup \perp \leftarrow_{\$} \textsc{Update}(E, c, w, z, \texttt{action})$, where $\texttt{action} \in \{\texttt{add}, \texttt{remove}\}$.

In addition to the standard Encapsulated Search Index properties of **Correctness**, **Uniqueness**, **CCA Security** and **Privacy-Preserving**, we require the following security properties from an updatable Encapsulated Search Index:

Update Correctness: over the randomness of KGen, Prep, Index, for any document D_0, update sequence $(\texttt{action}_1, w_1), \ldots, (\texttt{action}_q, w_q)$ and keyword w, the following holds with probability $(1 - \text{negl}(\kappa))$:

- Let $(c, E_0) \leftarrow \textsc{BldIdx}(PK, D_0)$.
- For $i = 1$ to q, let:
  - $z_i = \text{S-}\textsc{Prove}(SK, c, w_i)$;
  - $E_i = \textsc{Update}(E_{i-1}, c, w_i, z_i, \texttt{action}_i)$;
  - D_i be correct update of D_{i-1} following $\texttt{action}_i$ on w_i.
- Then $\textsc{Search}(SK, (E_q, c), w) = \begin{cases} 1 & \text{if } w \in D_q \\ 0 & \text{if } w \notin D_q \end{cases}$

8.3 Encapsulated Verifiable Random Functions (EVRFs)

As mentioned earlier, we use a new primitive called Encapsulated Verifiable Random Function to build the encapsulated search index. In this section, we begin by introducing this primitive in section 8.3.1. In Section 8.3.3, we present an overview of extensions to this primitive. Later sections in paper contained detailed expositions on the extensions.

8.3.1 Standard EVRFs

Intuitively, an EVRF allows the receiver Alice to publish a public key PK and keep secret key SK private so that any sender Bob can use PK to produce a ciphertext C and trapdoor key T in a way such that for any input x, the correct VRF value y on x can be efficiently evaluated in two different ways:

(a) Alice can evaluate y using secret key SK and ciphertext C.

(b) Bob can evaluate y using trapdoor T.

In addition, for any third party Charlie who knows C, PK and x:

(c) Alice can produce a proof z convincing Charlie that the value y is correct.

(d) Without such proof, the value y will look pseudorandom to Charlie.

Definition 8.10. An *Encapsulated Verifiable Random Function* (EVRF) is a tuple of PPT algorithms EVRF $=$ (Gen, Encap, Comp, Split, Core, Post) such that:

- $\text{Gen}(1^\kappa) \to (PK, SK)$: outputs the public/secret key pair.
- $\text{Encap}(PK) \to (C, T)$: outputs ciphertext C and trapdoor T.
- $\text{Comp}(T, x) = y$: evaluates EVRF on input x, using trapdoor T.

- $\textsc{Split}(PK, C') = R'$: outputs a handle from full ciphertext C'.

  Note, this preprocessing is independent of the input x, can depend on the public key PK, but *not* on the secret key SK.[5]

- $\textsc{Core}(SK, R', x) = z'$: evaluates partial EVRF output on input x, using the secret key SK and handle R'.

- $\textsc{Post}(PK, z', C', x) = y' \cup \perp$: outputs either the EVRF output from the partial output z', or $\perp$.

Before we define the security properties, it is useful to define the following shorthand functions:

- $\textsc{Prove}(PK, SK, C, x) = \textsc{Core}(SK, \textsc{Split}(PK, C), x)$

- $\mathsf{Eval}(PK, SK, C, x) = \textsc{Post}(PK, \textsc{Prove}(SK, C, x), C, x)$

We require the following security properties:

1. **Evaluation-Correctness**: with prob. 1 over randomness of $\textsc{Gen}$ and $\textsc{Encap}$, for honestly generated ciphertext C and for all inputs x,

$$\textsc{Comp}(T, x) = \mathsf{Eval}(PK, SK, C, x)$$

2. **Uniqueness**: there exist no values (PK, C, x, z_1, z_2) st$y_1 \neq \perp, y_2 \neq \perp,$ and $y_1 \neq y_2$ where

$$y_1 = \textsc{Post}(PK, z_1, C, x), \quad y_2 = \textsc{Post}(PK, z_2, C, x)$$

3. **Pseudorandomness under** $\textsc{Core}$ (**$-Core**): for any PPT algorithm $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, where

[5]The algorithm $\textsc{Split}$ is not technically needed, as one can always set $R = C$. In fact, this will be the case for our EVRF in section 8.3.4. However, one could envision EVRF constructions where the $\textsc{Split}$ procedure can do a non-trivial (input-independent) part of the overall $\textsc{Prove} = \textsc{Core}(\textsc{Split})$ procedure, and without the need to know the secret key SK. This will be the case for some of the delegatable EVRFs we consider in Section 8.5.1.

$\mathcal{A}$ does not make query (C, x) to $\textsc{Prove}(PK, SK, \cdot, \cdot)$, for variables SK, C, x defined below, the following holds:

$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} (PK, SK) \leftarrow\!\!\$\ \textsc{Gen}(1^\kappa); \\ (C, T) \leftarrow\!\!\$\ \textsc{Encap}(PK); \\ (x, st) \leftarrow\!\!\$\ \mathcal{A}_1^{\textsc{Prove}(PK,SK,\cdot,\cdot)}(PK, C); \\ y_0 = \textsc{Comp}(T, x);\ \ y_1 \leftarrow\!\!\$\ \{0,1\}^{|y_0|}; \\ b \leftarrow\!\!\$\ \{0,1\};\ \ b' \leftarrow\!\!\$\ \mathcal{A}_2^{\textsc{Prove}(PK,SK,\cdot,\cdot)}(y_b, st) \end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\kappa)$$

We present a construction of our EVRF in section 8.3.4.

Remark 7. We note that any valid ciphertext C implicitly defines a standard verifiable random function (VRF). In particular, the value $z = \textsc{Prove}(SK, C, x)$ could be viewed as the VRF proof, which is accepted iff $\textsc{Post}(PK, z, C, x) \neq \perp$.

Remark 8. We reiterate that our pseudorandomness definition does not give the attacker "unguarded" access to the Core procedure, but only "Split-guarded" access to $\textsc{Prove} = \textsc{Core}(\textsc{Split})$. This difference does not matter when the Split procedure just sets $R = C$. However, when Split is non-trivial, the owner of SK (Alice) can only outsource it to some outside server (Charlie) if it trusts Charlie and the authenticity (but not privacy) of the channel between Alice and Charlie.

8.3.2 Generic Construction of Encapsulated Search Index

Non-private Dictionary Data Structure. Our generic construction will use the simplest kind of non-cryptographic dictionary which allows one to preprocess some set D into some data structure E so that membership queries $w \in D$ can be answered in sub-linear time in $N = |D|$. In particular, a classic instantiation of such a dictionary could be any balanced search trees with search time $O(\log N)$. If a small probability of error is allowed, we could also use faster data structures, such as hash tables [CLRS09], Bloom filters [Blo70, PPR05, NY15] or cuckoo hash [PR04],

Construction: Generic ESI

KGen(1^κ)

Run $(PK, SK) \leftarrow_s$ EVRF.Gen(1^κ).
return PK, SK.

Prep(PK)

Run $(C, T) \leftarrow_s$ EVRF.Encap(PK).
return $c = C$ and $s = T$.

Index(s, D)

for $w \in D$ **do**
Compute $y_w =$ EVRF.Comp(s, w).
Compute $Y = \{y_w | w \in D\}$.
Run $E =$ DS.Construct(Y).
return E.

S-Split(PK, c')

Run EVRF.Split(PK, c') $= r'$.
return r'.

S-Core(SK, r', w)

Run EVRF.Core(SK, r', w) $= z'$.
return z'.

Finalize(PK, E', c', z', w)

Run EVRF.Post(PK, z', c', w) $= y'$.
if $y' = \perp$ **then**
return $\perp$.
else
return DS.Find(E', y').

Figure 8.2: Generic ESI = (KGen, Prep, Index, S-Split, S-Core, Finalize).

whose search takes expected time $O(1)$. The particular choice of the non-cryptographic dictionary will depend on the application, which is a nice luxury allowed by our generic composition.

Formally, a non-private dictionary DS = (Construct, Find) is any data structure supporting the following two operations:

- $E =$ Construct(D): outputs the index E on an input document D.
- Find(E, w) $\rightarrow \{0, 1\}$: outputs 1 if w is present in D, and 0 otherwise. We assume perfect correctness for $w \in D$, and allow negligible error probability for $w \notin D$.

Our Composition. We show that Encapsulated Search Index can be easily built from any such non-cryptographic dictionary DS = (Construct, Find) and and EVRF = (Gen, Encap, Comp, Split, Core, Post). This composition is given below in Construction 8.2.

Efficiency. By design, the Search operation of our composition inherits the efficiency of the non-cryptographic dictionary DS. In particular, it is $O(\log |D|)$ with standard balanced search trees and could become potentially $O(1)$ with probabilistic dictionaries, such as hash tables or Bloom filters.

Security Analysis. The **Correctness** and **Uniqueness** properties of the above construction trivially follows from the respective properties of the underlying EVRF and *DS*. In particular, we get negligible error probability for $w \notin D$ either due to unlikely EVRF collision between y_w and

$y_{w'}$ for some $w' \in D$, or a false positive of the DS.

Theorem 8.11. *If EVRF satisfies the **$-Core** property, then Encapsulated Search Index is **CCA** secure. Further, if the EVRF (resp. DS) is threshold and/or delegatable (resp. updatable; see Remark 9), the resulting ESI inherits the same.*

Proof. The proof is through a sequence of Hybrids where at each step we replace the index corresponding to a word $w \in (D_1 \setminus D_2) \cup (D_2 \setminus D_1)$ with a random index, one by one. Recall that the process of building an index is to use the keyword as the input to the EVRF, making the output of the EVRF as the index. The key idea behind the proof is that only these words can help distinguish the two documents and the adversary is prevented from receiving computation of S-Prove on these words. Thus, the EVRF of these words is never realized by the adversary. Let H_i be the distribution where the first i distinguishing words in D_1 have been replaced by a random string and call the resulting encrypted index E_i. We will show that if $\mathcal{A}$ can distinguish between H_i and H_{i+1}, then we can construct an adversary $\mathcal{B}$ that can win in the EVRF security game.

Formally, let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a PPT attacker trying to distinguish between hybrids H_i and H_{i+1}, having advantage ϵ. $\mathcal{A}$ is given public key PK (for unknown SK) and oracle access to S-Prove$(SK, \cdot, \cdot)$. In addition, the challenge index is either E_i or E_{i+1}.

Using this attacker $\mathcal{A}$, we now define a PPT attacker $\mathcal{B}$ which will break the **$−Core** property of $EVRF$. $\mathcal{B}$ is given the value PK, challenge ciphertext C, and oracle access to Prove. Note that S-Prove and Prove are identical functions for our construction.

Definition of $\mathcal{B}^{\text{Prove}(SK,\cdot,\cdot)}(PK, C)$.

- **Setup:** $\mathcal{B}$ provides to $\mathcal{A}$ the PK value.
- **Queries to S-Prove$(SK, \cdot, \cdot)$:** $\mathcal{B}$ receives inputs (c_i, w_i)
  - Uses its access to Prove queries to receive the output of Prove$(SK, c_i, w_i) = z_i$.
  - Responds to $\mathcal{A}$ with z_i after recording (c_i, w_i, z_i) in a table T.

- $\mathcal{B}$ uses table T to verify that $\mathcal{A}$ is not querying S-Prove on a distinguishing word w_i.

- **Challenge Query:** $\mathcal{B}$ receives two documents D_1, D_2. It identifies the set of distinguishing words W, i.e., $W = (D_1 \setminus D_2) \cup (D_2 \setminus D_1)$. Let $W = \{w'_1, \ldots, w'_m\}$. $\mathcal{B}$ tests the legality of $\mathcal{A}$ using T. If the test passes, then for $w \in D$, do the following:

  - If $w \in \{w_{i+2}, \ldots, w_m\}$ or $w \notin W$, then
    - Query oracle Prove with input (C, w) to receive as output $z = \text{Core}(SK, \text{Split}(PK, C), w)$
    - Compute $\text{Post}(PK, z, C, x) = y$.
    - Add y to Y.
  - If $w \in \{w_1, \ldots, w_i\}$, then:
    - Pick y at random.
    - Add y to Y,
  - If $w = w_{i+1}$, then:
    - $\mathcal{B}$ uses w_{i+1} as its challenge word. It receives as response y_{i+1}.
    - Add y_{i+1} to Y.

  Run DS.$\text{Construct}(Y) = E^*$. Set $c^* = C$. Forward (E^*, c^*) to $\mathcal{A}$.

- **Finish:** Let $\mathcal{A}$ return b' as its guess. It forwards b' as its guess.

Analysis of Reduction. If the challenger's bit was 0, then $\mathcal{B}$ simulates the distribution H_i perfectly, where the first i distinguishing words are indexed by a random value. If the challenger's bit was 1, then $\mathcal{B}$ simulates the distribution H_{i+1} perfectly where the first i+1 distinguishing words are indexed by a random value. Therefore, the advantage of $\mathcal{B}$ in distinguishing between real or random value is same as $\mathcal{A}$'s advantage in distinguishing between hybrids H_i and H_{i+1} (which is ϵ). □

Remark 9. It is easy to see that our construction of Encapsulated Search Index (Construction 8.2) can be made updatable if the underlying data structure supports the addition and removal. Formally the DS also has the following additional operation:

- DS.Modify$(E, w, \texttt{action}) \rightarrow E'$: adds/removes w to/from the index E when $\texttt{action} = \texttt{add/remove}$, and outputs the new index E' .

Then, the Update$(E, c, w, z, \texttt{action})$ algorithm can be defined as follows:

- Compute EVRF.Post$(PK, z, c, w) = y$.
- If $y = \perp$, return $\perp$.
- Otherwise, return $E' =$ DS.Modify$(E, y, \texttt{action})$

8.3.3 Extensions to EVRFs

The generic construction of ESI from Section 8.3.2 can be extended to achieve the various extensions of ESI, as defined in Section 8.2. We do this by extending the EVRF definitions, and instantiating each ESI with its corresponding EVRF and inheriting the required functionality.

Threshold EVRF. In the earlier definition, we had a single secret key SK. With possession of this secret key, one can evaluate the EVRF on any input x. Therefore, it becomes imperative to protect the key from leakage. Indeed, it is natural to extend our early definition to cater to the setting of a distributed evaluation of the EVRF. The key difference in the definition of threshold EVRF from the earlier definition is that the Post algorithm is now formally split into the share verification algorithm Shr-Vfy and the final evaluation algorithm Combine. The formal discussion about the syntax and security of this primitive can be found in Section 8.4.1.

Remark 10. It is easy to see that our construction of Encapsulated Search Index (Construction 8.2) inherits the different properties of the underlying encapsulated verifiable random func-

tion. In other words, by using TEVRF, one can construct a threshold Encapsulated Search Index by suitably mapping the functions as follows:

- KG-Verify$(PK, \mathbf{VK})$ = Gen-Vfy$(PK, \mathbf{VK})$
- S-Verify(sk_i, z'_i, w) = Shr-Vfy(sk_i, z'_i, w)
- S-Combine$(PK, E', c', z'_{i_1}, \ldots, z'_{i_t}, w)$: run

$$y' = \text{Combine}(PK, c', z'_{i_1}, \ldots, z'_{i_t}, w)$$

  and then output DS.Find(E', y')

Therefore, Theorem 8.11 can be extended to say that if TEVRF satisfies the **$-DCore** property, then we have a **CCA** secure threshold Encapsulated Search Index.

Delegatable EVRF. Next, we extend the definition of *standard* EVRFs to the setting where the EVRF owner could delegate its evaluation power to another key. Recall that a standard EVRF has the following algorithms: Gen, Encap, Comp, Split, Core, Post. Delegation, therefore, implies that one can convert a ciphertext C_1 for key pair (PK_1, SK_1) to ciphertext C_2 for a different key pair (PK_2, SK_2) which encapsulates the same VRF, i.e.,

$$\forall x,\ \mathsf{Eval}(PK_1, SK_1, C_1, x) = \mathsf{Eval}(PK_2, SK_2, C_2, x) \tag{8.2}$$

where $\mathsf{Eval}(PK, SK, C, x) = \text{Post}(PK, \text{Prove}(SK, c, x), C, x)$. The formal discussion about the syntax and security of this primitive can be found in Section 8.5.1.

Remark 11. It is easy to see that our construction of Encapsulated Search Index (Construction 8.2) inherits the different properties of the underlying encapsulated verifiable random function. In other words, by using a DEVRF, one can construct a delegatable Encapsulated Search Index by suitably mapping the functions as follows:

Construction: Standard EVRF

$\textsc{Gen}(1^\kappa)$

Sample $a \in_r \mathbb{Z}_p^*$
Compute $A = g^a \in G$.
return $SK = a$ and $PK = (g, A)$.

$\textsc{Encap}(PK)$

Parse $PK = (g, A)$.
Sample $r \in_r \mathbb{Z}_p^*$.
Compute $R = g^r, S = A^r$.
return $C = R$, $T = (R, S)$.

$\textsc{Comp}(T, x)$

Parse $T = (R, S)$.
Compute $y = e(H(R, x), S)$.
return y.

$\textsc{Split}(PK, C')$

Parse $PK = (g, A)$, $C' = R'$.
return R'.

$\textsc{Core}(SK, C', x)$

Parse $SK = a, C' = R'$.
Compute $z = H(R', x)^a$.
return z.

$\textsc{Post}(PK, z, C', x)$

Parse $PK = (g, A)$, $C' = R'$
if $e(z, g) \neq e(H(R', x), A)$ **then**
    return $\perp$.
else
    Compute $y' = e(z, R')$.
    return y'.

Figure 8.3: Standard EVRF = ($\textsc{Gen}$, $\textsc{Encap}$, $\textsc{Comp}$, $\textsc{Split}$, $\textsc{Core}$, $\textsc{Post}$).

- S-$\textsc{Del}(SK_1, c_1, SK_2) = \textsc{Del}(SK_1, c_1, SK_2)$
- S-$\textsc{Check}(PK_1, c_1, PK_2, c_2) = \textsc{Same}(PK_1, c_1, PK_2, c_2)$

In addition, the Encapsulated Search Index also achieves different levels of **CCA** security based on the security level of **$-Core** property of the delegatable EVRF.

8.3.4 Standard EVRF

We now present the standard EVRF construction, presented in Figure 8.3.

Security Analysis. To check **Evaluation-Correctness**, we observe that $A^r = g^{ar} = R^a$, and by the bilinearity we have:

$$\textsc{Comp}(T = (R, S), x) = e(H(R, x), S) = e(H(R, x), A^r)$$

From our earlier observation, we get that:

$$e(H(R, x), A^r) = e(H(R, x), R^a) = e(H(R, x)^a, R) = e(z, R)$$

This is the same as $\textsc{Post}(A, \textsc{Core}(a, \textsc{Split}(A, R), x), R, x)$ which concludes the proof.

To prove **Uniqueness**, consider any tuple $(PK = A, C = R, x, z_1, z_2)$. Further, let $y_1 = \text{Post}(A, z_1, R, x)$ and $y_2 = \text{Post}(A, z_2, R, x)$. If $y_1 \neq \perp$ and $y_2 \neq \perp$, then we have that $e(z_1, g) = e(H(R, x), A) = e(z_2, g)$. From definition of bilinear groups, we get that $z_1 = z_2$. Consequently, $y_1 = e(z_1, R) = e(z_2, R) = y_2$.

Theorem 8.12. *The standard EVRF given in Figure 8.3 satisfies the* ***$-Core*** *property under the* BDDH *assumption in the random oracle model.*

Proof. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a PPT attacker against the **$-Core** property of EVRF, having advantage ϵ. $\mathcal{A}$ is given the public key $A = g^a$ (for unknown a), challenge ciphertext $R = g^r$ (for unknown r), and oracle access to Prove, which is equal to Core due to empty Split step. Namely, $\mathcal{A}$ has access to $\text{Core}(a, \cdot, \cdot)$: on query (R', x'), $\mathcal{A}$ gets $z' = H(R', x')^a$. $\mathcal{A}$ then outputs challenge x, gets back a value y which is either $e(H(R, x)^a, R) = e(H(R, x), g)^{ar}$ or uniform over G_1, and has to tell which without asking its $\text{Core}(a, \cdot, \cdot)$ oracle on the challenge (R, x). Additionally, $\mathcal{A}$ expects to have oracle access to the random oracle $H : \{0,1\}^* \to G$, and values $g, A = g^a, B = g^b, R = g^r$ (for unknown a, b, r), and a challenge g_1, which is either $e(g, g)^{abr}$ or uniform in G_1. There is no random oracle for $\mathcal{B}$.

Definition of $\mathcal{B}(g, A, B, R, g_1)$. Run $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ as follows:

- **Setup:** Pass $PK = A$ and challenge ciphertext $C = R$ to $\mathcal{A}_1$.
- **Queries to H:** let q be the upper bound on the number of queries to Core made by $\mathcal{A}$. $\mathcal{B}$ will maintain a table T — initially empty — of the form $\{((R_i, x_i), \beta_i, coin_i)\}$, where i is the index of the query to H incremented with each query, (R_i, x_i) in the input to the query (inputs not of this form are consistently answered at random), and the meaning of $\beta_i \in \mathbb{Z}_p$ and $coin_i \in \{0, 1\}$ will be explained below:

  1. If (R_i, x_i) was already made, meaning there is $j < i$ such that $(R_i, x_i) = (R_j, x_j)$ and $((R_j, x_j), \beta_j, coin_j) \in T$, then respond with:

$H(R_i, x_i) = g^{\beta_j}$ if $coin_j = 0$, and with $H(R_i, x_i) = B^{\beta_j}$ if $coin_j = 1$.

2. Otherwise, flip a fresh $coin_i \in \{0, 1\}$ such that $\Pr[[]coin_i = 1] = 1/q$, sample random $\beta_i \leftarrow_\$ \mathbb{Z}_p$, add the tuple $((R_i, x_i), \beta_i, coin_i)$ to T, and respond with:

$H(R_i, x_i) = g^{\beta_i}$ if $coin_i = 0$, and with $H(R_i, x_i) = B^{\beta_i}$ if $coin_i = 1$.

- **Queries to** CORE**:** Given query (R', x') to CORE, check if $\mathcal{A}$ made the H-query (R', x') by checking if there exists j such that $(R', x') = (R_j, x_j)$ and $((R_j, x_j), \beta_j, coin_j) \in T$. If such query was not made, make this query for $\mathcal{A}$ following the normal H-query simulation strategy above. In either case, retrieve the record $((R_j, x_j), \beta_j, coin_j) \in T$ with $(R', x') = (R_j, x_j)$. Respond as follows:

  If $coin_j = 1$, abort the simulation, outputting a random bit $b' \in_r \{0, 1\}$.

  Otherwise, we know $H(R', x') = g^{\beta_j}$, so $\mathcal{B}$ outputs partial output $z' = A^{\beta_j}$.

  (Notice, $z' = A^{\beta_j} = g^{a\beta_j} = H(R', x')^a$, for unknown a.)

- **Challenge query:** When $\mathcal{A}_1$ produces challenge x, produce answer y as follows. Check if $\mathcal{A}_1$ made the H-query (R, x) by checking if there exists j such that $(R, x) = (R_j, x_j)$ and $((R_j, x_j), \beta_j, coin_j) \in T$. If such query was not made, make this query for $\mathcal{A}$ following the normal H-query simulation strategy above. In either case, retrieve the record $((R_j, x_j), \beta_j, coin_j) \in T$ with $(R, x) = (R_j, x_j)$. Respond as follows:

  If $coin_j = 0$, abort the simulation, outputting a random bit $b' \in_r \{0, 1\}$.

  Otherwise, we know $H(R, x) = B^{\beta_j}$, and $\mathcal{B}$ will output challenge $y = g_1^{\beta_j}$.

(Notice, $y_0 = \mathsf{Eval}(a, R, x) = e(H(R,x)^a, R) = e(B^{\beta_j a}, g^r) = (e(g,g)^{abr})^{\beta_j}$.)

- **Finish:** If the simulation did not fail, and $\mathcal{A}$ produces a guess b', $\mathcal{B}$ outputs the same b'.

Analysis of the Reduction. Assume the advantage of $\mathcal{A}$ is ϵ, and let us denote by ϵ' the advantage of $\mathcal{B}$. We will argue that $\epsilon' > \epsilon/3q$, which would complete the proof, since q is polynomial. Let use define "failure event" F to be that $\mathcal{B}$ had to abort the simulation of $\mathcal{A}$, either during one of the Core queries (if $coin_j = 1$), or when simulating the challenge (if $coin_j = 0$). Recall, if F happens, $\mathcal{B}$ still outputs a random bit b', which has $1/2$ chance to be equal to the challenge b. Let us also define "success probability" $\gamma \in [0,1]$ as $\Pr[\,]\neg F] = \gamma$. Notice,

$$\begin{aligned}
\frac{1}{2} + \epsilon' &= \Pr[\,]b = b'] \\
&= \Pr[\,]F] \cdot \Pr[\,]b = b' \mid F] + \Pr[\,]\neg F] \cdot \Pr[\,]b = b' \mid \neg F] \\
&= (1-\gamma) \cdot \frac{1}{2} + \gamma \cdot \Pr[\,]b = b' \mid \neg F] \\
&= \frac{1}{2} + \gamma \cdot \left(\Pr[\,]b = b' \mid \neg F] - \frac{1}{2} \right)
\end{aligned}$$

Thus, to complete our proof that $\epsilon' > \epsilon/3q$, it suffices to show the following two claims:

Claim 8.13. $\gamma = \Pr[\,]\neg F] > \frac{1}{3q}$.

Claim 8.14 (2). $\Pr[\,]b = b' \mid \neg F] = \frac{1}{2} + \epsilon$.

Proof of Claim 8.13. We see that in order for $\mathcal{B}$ not to abort the simulation, all random bits $coin_j$ have to be equal to 0 (each independently happens with probability $1 - 1/q$) when simulating at most q Core queries, and also the "challenge" coin $coin_j$ should be 1 (independently happens with probability $1/q$, since none of the Core queries used the challenge (R, x)). This means that, overall,

$$\gamma \geqslant \left(1 - \frac{1}{q}\right)^q \cdot \frac{1}{q} > \frac{1}{3q}$$

completing the proof. □

Proof of Claim 8.14. We claim that when $\mathcal{B}$ successfully completes the simulation, he perfectly simulates the run of $\mathcal{A}$. More precisely, in this case the run of $\mathcal{B}$ with challenge bit $b = 0$ is identical to the run of $\mathcal{A}$ with challenge bit $b = 0$, and the same for $b = 1$. This will complete the proof, as then

$$\Pr[[]b = b' \mid \neg F] = \Pr[[]\mathcal{A} \text{ wins}] = \frac{1}{2} + \epsilon$$

To see this, first we notice that all the queries to H are answered at random irrespective of the value of the $coin_j$, since both distributions g^{β_j} and B^{β_j} are perfectly uniform when $\beta_j \leftarrow_\$ \mathbb{Z}_p$.

Second, we already saw that when all Core queries are answered, each partial answer z' is correct, as $z' = A^{\beta_j} = g^{a\beta_j} = H(R', x')^a$ (for correct, although unknown, a). Finally, let use look at the challenge query (R, x). Since the simulation succeeded, we know $H(R, x) = B^{\beta_j}$, for some fresh and random $\beta_j \in \mathbb{Z}_p$. This means the correct output $y_0 = \text{Eval}(a, R, x) = e(H(R, x)^a, R) = e(B^{\beta_j a}, g^r) = (e(g, g)^{abr})^{\beta_j}$. Our simulator $\mathcal{B}$ responded with $g_1^{\beta_j}$, where g_1 is its own challenge. Thus:

When $b = 0$, we have $g_1 = e(g, g)^{abr}$, meaning that $y_0 = g_1^{\beta_j}$ indeed.

When $b = 1$, $g_1 \in_r G_1$, which means that the response $g_1^{\beta_j}$ is identically distributed with a uniform answer $y_1 \in_r G_1$, as needed.

This completes the proof of Claim 8.13 and Theorem 8.12. □

8.4 Threshold Encapsulated Verifiable Random Functions

In this section, we formally introduce the primitive known as a Threshold EVRF in Section 8.4.1. We then present a construction of Threshold EVRF in Section 8.4.2 .

8.4.1 Definition of Threshold (or Distributed) EVRFs

Definition 8.15. A (t, n)-*Threshold EVRF* is a tuple of PPT algorithms TEVRF = (Gen, Gen-Vfy, Encap, Comp, Split, D-Core, Shr-Vfy, Combine) such that:

- $(PK, \mathbf{SK} = (sk_1, \ldots, sk_n), \mathbf{VK} = (vk_1, \ldots, vk_n)) \leftarrow\!\!\$\ \textsc{Gen}(1^\kappa, t, n)$: outputs the public key PK, a vector of secret shares $\mathbf{SK}$, and public shares $\mathbf{VK}$.
- $\textsc{Gen-Vfy}(PK, \mathbf{VK}) = \beta \in \{0, 1\}$: verifies that the output of Gen is indeed valid.
- $(C, T) \leftarrow\!\!\$\ \textsc{Encap}(PK)$: outputs ciphertext C and trapdoor T.
- $\textsc{Comp}(T, x) = y$: evaluates EVRF on input x, using trapdoor T.
- $\textsc{Split}(PK, n, C') = (R'_1, \ldots R'_n)$: outputs n handles $R'_1, \ldots, R'_n$ from full ciphertext C'.
- $\textsc{D-Core}(sk_i, R'_i, x) = z'_i$: evaluates EVRF share on input x, using handle R'_i and secret key share sk_i.
- $\textsc{Shr-Vfy}(PK, vk_i, z'_i, x) = \beta \in \{0, 1\}$: verifies that the share produced by the party i is valid.
- $\textsc{Combine}(PK, C', z'_{i_1}, \ldots, z'_{i_t}, x) = y'$: uses the partial evaluations $z'_{i_1}, \ldots, z'_{i_t}$ to compute the final value of EVRF on input x.[6]

Before we define the security properties, it is useful to define the following shorthand functions:

- $\textsc{Prove}(\mathbf{SK}, i, C, x) = \textsc{D-Core}(sk_i, R_i, x)$, where $(R_1, \ldots, R_n) = \textsc{Split}(PK, n, C)$.

[6] Without loss of generality, we will always assume that all the t partial evaluations z'_i satisfy $\textsc{Shr-Vfy}(PK, vk_i, z'_i) = 1$ (else, we output $\perp$ before calling Combine). See also the definition of Eval below to explicitly model this assumption.

- $\mathsf{Eval}(\mathbf{SK}, i_1, \ldots, i_t, C, x)$: For $j = 1 \ldots t$, compute $z_{i_j} = \textsc{Prove}(\mathbf{SK}, i_j, C, x)$.

  Output $\perp$ if, for some $1 \leqslant j \leqslant t$, $\textsc{Shr-Vfy}(PK, vk_{i_j}, z_{i_j}, x) = 0$.

  Otherwise, output $\textsc{Combine}(PK, C, z_{i_1}, \ldots, z_{i_t}, x)$.

We require the following security properties:

1. **Distribution-Correctness**:

   (a) with prob. 1 over randomness of $(PK, \mathbf{SK}, \mathbf{VK}) \leftarrow\!\!\$\ \textsc{Gen}(1^\kappa, t, n)$,

   $\textsc{Gen-Vfy}(PK, \mathbf{VK}) = 1$

   (b) with prob. 1 over randomness of Gen and Encap, for honestly generated ciphertext C: $\mathsf{Eval}(\mathbf{SK}, i_1, \ldots, i_t, C, x) = \textsc{Comp}(T, x)$

2. **Uniqueness**: there exists no values $(PK, \mathbf{VK}, C, x, Z_1, Z_2)$ where $Z_1 = ((i_1, z_{i_1}), \ldots, (i_t, z_{i_t}))$ and $Z_2 = ((j_1, z_{j_1}), \ldots, (j_t, z_{j_t}))$. st

   (a) $\textsc{Gen-Vfy}(PK, \mathbf{VK}) = 1$

   (b) for $k = 1, \ldots, t$:

   - $\textsc{Shr-Vfy}(PK, vk_{i_k}, z_{i_k}, x) = 1$.
   - $\textsc{Shr-Vfy}(PK, vk_{j_k}, z_{j_k}, x) = 1$.

   (c) Let $\mathbf{Z}_i = (z_{i_1}, \ldots, z_{i_t})$ and $\mathbf{Z}_j = (z_{j_1}, \ldots, z_{j_t})$. Then,

   $$\textsc{Combine}(PK, C, \mathbf{Z}_i, x) \neq \textsc{Combine}(PK, C, \mathbf{Z}_j, x)$$

3. **Pseudorandomness under D-Core ($-DCore)**: for any PPT algorithm $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$, where $\mathcal{A}$ does not make query (j, C, x) to $\textsc{Prove}(\mathbf{SK}, \cdot, \cdot, \cdot)$, for $j \notin \{i_1, \ldots, i_{t-1}\}$ for

variables $i_1, \ldots, i_{t-1}, \mathbf{SK}, C, x$ defined below,

$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} \{i_1, \ldots, i_{t-1}, st\} \leftarrow \mathcal{A}_0(1^\kappa, t, n); \\ (PK, \mathbf{SK}, \mathbf{VK}) \leftarrow\!\!\$\ \textsc{Gen}(1^\kappa, t, n); \\ (C, T) \leftarrow\!\!\$\ \textsc{Encap}(PK); \\ (R_1, \ldots, R_n) = \textsc{Split}(PK, n, C); \\ (x, st) \leftarrow\!\!\$\ \mathcal{A}_1^{\textsc{Prove}(\mathbf{SK}, \cdot, \cdot, \cdot)}(PK, C, \mathbf{VK}, \mathbf{SK}', st) \\ y_0 = \textsc{Comp}(T, x); \quad y_1 \leftarrow\!\!\$\ \{0,1\}^{|y_0|}; \\ b \leftarrow\!\!\$\ \{0,1\}; \quad b' \leftarrow\!\!\$\ \mathcal{A}_2^{\textsc{Prove}(SK, \cdot, \cdot)}(y_b, st) \end{array} \right] \leq \frac{1}{2} + \text{negl}(\kappa)$$

where $\mathbf{SK}' = (sk_{i_1}, \ldots, sk_{i_{t-1}})$.

We present a construction of our threshold EVRF in section 8.4.2.

Remark 12. For simplicity, in the above definition, we assume honest key generation and do not explicitly address distributed key generation. Even with this simplification, the existence of the Gen-Vfy algorithm ensures the users of the system that the public key $(PK, \mathbf{VK})$ is "consistent "and was generated properly. Moreover, our construction, given in Section 8.4.2, can easily achieve efficient distributed key generation using techniques of Gennaro *et al.*[GJKR07].

Remark 13. Note that when $t = n = 1$, our threshold EVRF implies the the standard EVRF definition (Definition 8.10), where Post algorithm first runs Shr-Vfy on the single share z and then, if successful, runs Combine to produce the final output y. For $n > 1$, however, we find it extremely convenient that we can separately check the validity of each share, and be guaranteed to compute the correct output the moment t servers return consistent (i.e., Shr-Vfy'ed) shares z_i.

8.4.2 Construction of Threshold (or Distributed) EVRFs

Our non-interactive threshold EVRF is given in Figure 8.4. It combines elements of our standard EVRF from Figure 8.3 with the ideas of Shamir's Secret Sharing [Sha79], Feldman VSS [Fel88],

Construction: Non-Interactive Threshold EVRF

$\underline{\textsc{Gen}(1^\kappa)}$

Sample a random $(t-1)$ degree polynomial $f \in \mathbb{Z}_p^*[X]$.
Compute $a = f(0)$, $A_0 = g^a$.
for $i = 1, \ldots, n$ **do**
  Compute $a_i = f(i), A_i = g^{a_i}$.
return $PK = (g, A_0)$, $\mathbf{SK} = (a_1, \ldots, a_n)$, $\mathbf{VK} = (A_1, \ldots, A_n)$,
with server i getting secret key $sk_i = a_i$ and verification key $vk_i = A_i$.

$\underline{\textsc{Gen-Vfy}(PK, \mathbf{VK})}$

Parse $PK = (g, A_0), \mathbf{VK} =, (A_1, \ldots, A_n))$.
for $i = t, \ldots, n$ **do**
  Compute Lagrange coefficients $\lambda_{i,0} \ldots, \lambda_{i,t-1}$ s.t. $f(i) = \sum_{j=0}^{t-1} \lambda_{i,j} \cdot f(j)$.
  Each $\lambda_{i,j}$ is a fixed constant.
  if $A_i \neq \prod_{j=0}^{t-1} A_j^{\lambda_{i,j}}$ **then**
    return 0
return 1

$\underline{\textsc{Encap}(PK)}$

Parse $PK = (g, A_0)$.
Sample $r \in_r \mathbb{Z}_p^*$.
Compute $R = g^r, S = A_0^r$.
return ciphertext $C = R$ and trapdoor $T = (R, S)$.

$\underline{\textsc{Comp}(T, x)}$

Parse $T = (R, S)$.
Compute $y = e(H(R, x), S)$.
return y.

$\underline{\textsc{Split}(PK, C')}$

Parse $PK = (g, A_0)$, $C' = R'$.
return $R'_1 = R', \ldots, R'_n = R'$.

$\underline{\textsc{D-Core}(SK_i, R'_i, x)}$

Parse $SK_i = a_i, R'_i = R'$.
Compute partial output $z_i = H(R'_i, x)^{a_i}$.
return z_i.

$\underline{\textsc{Shr-Vfy}(PK, VK_i, z'_i, x)}$

Parse $PK = (g, A_0), VK_i = A_i$.
if $e(z'_i, g) \neq e(H(R'_i, x), A_i)$ **then**
  return $\perp$.

$\underline{\textsc{Combine}(PK, C', z'_{i_1}, \ldots, z'_{i_t}, x)}$

Parse $PK = (g, A_0)$, $C' = R'$.
Compute Lagrange coefficients $\lambda_1 \ldots, \lambda_t$ s.t. $f(0) = \sum_{j=1}^{t} \lambda_j \cdot f(i_j)$.
Note that these λ_j's only depend on indices $i_1, \ldots, i_t$.
Compute $z' = \prod_{j=1}^{t} (z'_{i_j})^{\lambda_j}$.
return $y = e(z', R')$.

Figure 8.4: TEVRF = (Gen, Gen-Vfy, Encap, Comp, Split, D-Core, Shr-Vfy, Combine).

and the fact that the correctness of all computations is easily verified using the pairing.

Security Analysis. To check **Distribution-Correctness**, we observe that $A = g^a$, $S = g^{ar}$, and $R = g^r$. Therefore, $\textsc{Comp}(T = (R, S), x) = e(H(R, x), S) = e(H(R, x), g)^{ar}$. By definition, we have that:

$$\mathsf{Eval}(PK, \mathbf{SK}, i_1, \ldots, i_t, R, x) = e(\prod_{j=1}^{t} z_{i_j}^{\lambda_j}, R)$$

$$e(\prod_{j=1}^{t} z_{i_j}^{\lambda_j}, R) = e(\prod_{j=1}^{t} H(R, x)^{a_{i_j} \cdot \lambda_j}, R) = e(H(R, x)^{\sum_{j=1}^{t} a_{i_j} \cdot \lambda_j}, R)$$

However, we know that $a = \sum_{j=1}^{t} a_{i_j} \cdot \lambda_j$. Therefore,

$$e(H(R,x)^{\sum_{j=1}^{t} a_{i_j} \cdot \lambda_j}, R) = e(H(R,x)^a, g^r) = e(H(R,x), g)^{ar}$$

To check **Uniqueness**, we are given: $(PK, \mathbf{VK} = (vk_1, \ldots, vk_n), R, x, Z_1, Z_2)$ where

$$Z_1 = ((i_1, z_{i_1}), \ldots, (i_t, z_{i_t})); \; Z_2 = ((j_1, z_{j_1}), \ldots, (j_t, z_{j_t})).$$

- $\textsc{Gen-Vfy}(PK, \mathbf{VK}) = 1$ implies that $a_0, a_1, \ldots, a_n$ where $g^{a_0} = PK$ and $g^{a_i} = vk_i$ all lie on a consistent polynomial f of degree $t-1$. Thus, there exist $\lambda_1, \ldots, \lambda_t \in \mathbb{Z}_p$ such that $f(0) = \sum_{\ell=1}^{t} \lambda_\ell \cdot f(i_\ell)$ and $\lambda'_1, \ldots, \lambda'_t \in \mathbb{Z}_p$ such that $f(0) = \sum_{\ell=1}^{t} \lambda_\ell \cdot f(j_\ell)$. Therefore, we have that:

$$A = \prod_{\ell=1}^{t} {vk_{i_\ell}}^{\lambda_\ell} = \prod_{\ell=1}^{t} {vk_{j_\ell}}^{\lambda'_\ell} \tag{8.3}$$

- We also know that for $\ell = 1, \ldots, t$,
  $\textsc{Shr-Vfy}(PK, vk_{i_\ell}, z_{i_\ell}, x) = 1$ and $\textsc{Shr-Vfy}(PK, vk_{j_\ell}, z_{j_\ell}, x) = 1$. Therefore, we have that for $\ell = 1, \ldots, t$:

$$e(z_{i_\ell}, g) = e(H(R,x), vk_{i_\ell}); \; e(z_{j_\ell}, g) = e(H(R,x), vk_{j_\ell}) \tag{8.4}$$

- We will now show that the 2 outputs of $\textsc{Combine}$ must be equal. Here we we will write $R = g^r$ for some r,

$$\textsc{Combine}(PK, R, z_{i_1}, \ldots, z_{i_t}, x) = e(\prod_{\ell=1}^{t} z_{i_\ell}^{\lambda_\ell}, R) = \prod_{\ell=1}^{t} e(z_{i_\ell}, g)^{r \cdot \lambda_\ell}$$

From Equation (8.4):

$$\prod_{\ell=1}^{t} e(z_{i_\ell}, g)^{r\cdot\lambda_\ell} = \prod_{\ell=1}^{t} e(H(R,x), vk_{i_\ell})^{r\cdot\lambda_\ell} = e(H(R,x), \prod_{\ell=1}^{t} vk_{i_\ell}^{\lambda_\ell})^r$$

From Equation (8.3), we have that:

$$e(H(R,x), \prod_{\ell=1}^{t} vk_{i_\ell}^{\lambda_\ell})^r = e(H(R,x), \prod_{\ell=1}^{t} vk_{j_\ell}^{\lambda'_\ell})^r = \prod_{\ell=1}^{t} e(H(R,x), vk_{j_\ell})^{r\cdot\lambda'_\ell}$$

We again use Equation (8.4) to conclude the proof.

Theorem 8.16. *If Figure 8.3 satisfies the **$-Core** property of standard EVRF, then Figure 8.4 satisfies the **$-DCore** property of threshold EVRF. By Theorem 8.12, it follows that Figure 8.4 satisfies the **$-DCore** property under the* BDDH *assumption in the random oracle model.*

Proof. Let $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$ be a PPT attacker against the **$-DCore** property of TEVRF, having advantage ϵ. $\mathcal{A}$ first chooses $t-1$ indices $S = \{i_1, \ldots, i_{t-1}\}$ where each index is a subset of $\{1, \ldots, n\}$. Then, $\mathcal{A}$ is given the public key $A = g^a$ (for unknown a), challenge ciphertext $R = g^r$ (for unknown r), secret keys $sk_{i_1} = a_{i_1} = f(i_1), \ldots, sk_{i_{t-1}} = a_{i_{t-1}} = f(i_{t-1})$ (for unknown polynomial f of degree t such that $f(0) = a$), verification keys **VK**, and oracle access to Prove, with equal valued Split step. Namely, $\mathcal{A}$ has access to $\textsc{Prove}(sk_i, \cdot, \cdot)$: on query (i, R', x'), $\mathcal{A}$ gets $z' = H(R', x')^{a_i}$. $\mathcal{A}$ then outputs challenge x, gets back a value y which is either $e(H(R,x)^a, R) = e(H(R,x), g)^{ar}$ or uniform over G_1, and has to tell which without asking its $\textsc{Prove}(\mathbf{SK}, \cdot, \cdot, \cdot)$ oracle on input (j, R, x) for $j \notin S$. Additionally, $\mathcal{A}$ expects to have oracle access to the random oracle $H : \{0,1\}^* \to G$.

Using this attacker $\mathcal{A}$, we now define a PPT attacker $\mathcal{B}$ which will break the **$-Core** property of EVRF. $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ is given the values g, public key $A^* = g^{a^*}$ (for unknown a^*), ciphertext $R^* = g^{r^*}$ (for unknown r^*), and an oracle access to Prove which is equal to Core due to empty Split step. Namely $\mathcal{B}$ has access to $\textsc{Core}(a^*, \cdot, \cdot)$ on query (R', x'), it receives in

response $H(R', x')^{a^*}$. $\mathcal{B}$ then outputs a challenge x and receives a response y which is either $e(H(R^*, x)^{a^*}, R^*) = e(H(R^*, x), g)^{a^* r}$ or uniform in G_1. Additionally, $\mathcal{B}$ expects to have oracle access to the random oracle $H : \{0,1\}^* \to G$.

Definition of $\mathcal{B}^{\text{Core}(a^*,\cdot,\cdot),H(\cdot,\cdot)}(A^*, R^*)$. Run $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$ as follows:

- **Setup:** $\mathcal{B}$ does the following during Setup.
  - Receive set $S = \{i_1, \ldots, i_{t-1}\}$ from $\mathcal{A}$.
  - Next $\mathcal{B}$ generates the key shares and public key as follows:
    * Sample $a_{i_1}, \ldots, a_{i_{t-1}} \in \mathbb{Z}_p$.
    * Pick $i_t \notin S$ at random.
    * Let $f \in \mathbb{Z}_p[X]$ be the degree $t-1$ polynomial implicitly defined to satisfy $f(i_t) = a^*$, and $f(i_j) = a_{i_j}$ for $j = 1, \ldots, t-1$.
    * Note that $\mathcal{B}$ does not know f since it does not know a^*.
    * Recall that $f(0) = a$ and $A = g^a$ is the public key. $\mathcal{B}$ determines the Lagrange coefficients $\lambda_1, \ldots, \lambda_{t-1}, \lambda_t \in \mathbb{Z}_p$ such that $f(0) = \sum_{j=1}^{t} \lambda_j \cdot f(i_j)$. Note that these do not require the knowledge of f. Therefore, we can now compute $A = \prod_{j=1}^{t-1} g^{a_{i_j}\lambda_j} \cdot (A^*)^{\lambda_t}$.
    * $\mathcal{B}$ gives $a_{i_1}, \ldots, a_{i_{t-1}}, A$ to $\mathcal{A}$.
  - Next $\mathcal{B}$ computes the verification key $\mathbf{VK} = (vk_1, \ldots, vk_n)$ as follows:
    * For $i_j \in S$, this is easy and merely sets $vk_{i_j} = g^{a_{i_j}}$. Further it sets $vk_{i_t} = A^*$.
    * For $\ell \notin S$, $\mathcal{B}$ determines the Lagrange coefficients $\lambda'_1, \ldots, \lambda'_t$ such that $f(j) = \sum_{j=1}^{t} \lambda_j \cdot f(i_j)$. Again, this does not need knowledge of f. Now, it can set $vk_\ell = \prod_{j=1}^{t} vk_{i_j}^{\lambda'_j}$.
    * $\mathcal{B}$ gives $\mathbf{VK}$ to $\mathcal{A}$.
  - $\mathcal{B}$ also provides R^* to $\mathcal{A}$.

- **Queries to H:** $\mathcal{B}$ merely responds to all queries from $\mathcal{A}$ to H by using its oracle access to H.

- **Queries to Prove:** $\mathcal{B}$ will maintain a table T, which is initially empty, of the form $\{(j_k, R_k, x_k), w_k\}$ where k is the index of the query to Prove incremented with each query (j_k, R_k, x_k). On receiving a query (j_k, R_k, x_k), it does the following:

  - If there exists $\ell < k$ such that $j_k = j_\ell, R_k = R_\ell, x_k = x_\ell$, then set $w_k = w_\ell$.
  - Else if $j_k \in S$, then it merely uses its oracle access to H to receive the value $H(R_k, x_k) = h_k$. It then sets $w_k = h_k^{a_{j_k}}$.
  - Else if $j_k = i_t$, then it uses it oracle access to Core to receive w_k which is actually equal to $H(R_k, x_k)^{a^*}$.
  - For all other j_k, it does the following:
    * Uses its oracle access to H to receive the value $h_k = H(R_k, x_k)$.
    * Further uses its oracle access to Core to receive the value h^* which is actually $H(R_k, x_k)^{a^*}$.
    * Determines Lagrange coefficients $\lambda_1, \ldots, \lambda_t$ such that $f(j_k) = \sum_{\ell=1}^{t} \lambda_\ell \cdot f(i_\ell)$. Now, it computes $w_k = (h^*)^{\lambda_t} \cdot \prod_{\ell=1}^{t-1} h_k^{a_{i_\ell} \cdot \lambda_\ell}$
  - Record $((j_k, R_k, x_k), w_k)$ and return w_k to $\mathcal{A}$.

- **Challenge Query:** On receiving the challenge input x, $\mathcal{B}$ does the following:

  - Check if there exists $((j_k, R_k, x_k), w_k) \in T$ such that $j_k \notin S$, $R_k = R^*$, and $x_k = x$. If yes, abort as $\mathcal{A}$ has violated the definition of the game.
  - If not, $\mathcal{B}$ issues its challenge input as x. It receives y^* which is either $e(H(R^*, x), R^*)^{a^*}$ or a random element in $\mathbb{G}_1$.
  - It also uses its oracle access to H to receive $h = H(R^*, x)$.

- Determines Lagrange coefficients $\lambda_1, \ldots, \lambda_t$ such that $f(0) = \sum_{\ell=1}^{t} \lambda_\ell \cdot f(i_\ell)$.
- Computes $z^* = \prod_{\ell=1}^{t-1} h^{\lambda_\ell \cdot a_{i_\ell}}$.
- It finally computes $y = y^{*\lambda_t} \cdot e(z^*, R^*)$ and outputs y to $\mathcal{A}$

- **Finish:** It forwards $\mathcal{A}$'s guess as its own guess.

Analysis of the Reduction. When $y^* = e(H(R^*, x), R^*)^{a^*}$, then we get that:

$$\begin{aligned}
y &= y^{*\lambda_t} \cdot e(z^*, R^*) \\
&= e(H(R^*, x), R^*)^{a^* \cdot \lambda_t} \cdot e(z^*, R^*) \\
&= e(h, R^*)^{a^* \cdot \lambda_t} \cdot e(\prod_{\ell=1}^{t-1} h^{\lambda_\ell \cdot a_{i_\ell}}, R^*) \\
&= e(h, R^*)^{f(i_t) \cdot \lambda_t} \cdot e(h^{\sum_{\ell=1}^{t-1} \lambda_\ell \cdot f(i_\ell)}, R^*) \\
&= e(h, R^*)^{\sum_{\ell=1}^{t} \lambda_\ell \cdot f(i_\ell)} = e(h, R^*)^{f(0)} = e(h, R^*)^{a}
\end{aligned}$$

It is easy to see that if y^* was a random group element in $\mathbb{G}_1$, the final output y is also a random group element. Therefore, $\mathcal{B}$ perfectly simulates the **$ – Core** game for $\mathcal{A}$, and $\mathcal{B}$'s advantage is the same as $\mathcal{A}$'s advantage. This completes the proof of Theorem 8.16. □

8.5 Delegatable Encapsulated Verifiable Random Functions

In this section, we formally introduce the primitive known as a Delegatable EVRF in Section 8.5.1. This definition captures different levels of delegatability and we present constructions that satisfy these levels in Sections 8.5.2,8.5.3, and 8.5.4.

8.5.1 Definition of Delegatable EVRFs

In this work, we will be interested in a stronger type of delegatable EVRFs where anybody can check if two ciphertexts C_1 and C_2 "came from the same place". This is governed by the "comparison" procedure $\textsc{Same}(PK_1, C_1, PK_2, C_2)$ which outputs 1 only if Equation (8.2) holds. This procedure will have several uses. First, it allows the owner of SK_2 to be sure that the resulting ciphertext C_2 indeed encapsulates the same VRF under PK_2 as C_1 does under PK_1. Second, it will allow us to cleanly define a "trivial" attack on the pseudorandomness of delegatable EVRFs. See also Remark 15.

Before we define the syntax and the security of delegatable EVRFs, we include a brief exposition on the nuances in the syntax and security of such a primitive.

Secretly-Delegatable vs Publicly-Delegatable EVRFs. It is fairly obvious from the security of EVRFs that the delegation procedure must use the secret key SK_1 of the delegating party. The big distinction/subtlety comes from whether or not such delegation also requires the secret key SK_2 of the receiving party. In our basic notion, defined below, we will allow such a dependence. However, for completeness, we also define *publicly-delegatable* EVRFs, where only the public key PK_2 is needed. A priori, publicly delegatable EVRFs have the advantage that the delegation does not need the cooperation of the receiving party. However, all our secretly-delegatable schemes will have a trivial implementation, where the sender can use SK_1 to non-interactively convert C_1 into the "C_1-specific" trapdoor T_1 of the EVRF, which it can (securely) send to the receiving party. In turn, the receiving party can use the secret key SK_2 to convert T_1 to the corresponding delegated ciphertext C_2. Thus, all our concrete secret-key delegatable EVRFs will have no "usability disadvantages" compared to publicly-delegatable EVRFs.

Bounded Delegation. Additionally, the definition we present is for unbounded delegation, where any delegated ciphertext can be further delegated. We could also restrict the definition to *t-delegatable*, with $t = 1$ being an important special case, where **Delegation-Completeness** is

only required to hold for up to t iterated delegations starting from any ciphertext C_1 output by the encapsulation procedure Encap. We will consider such a variant in Section 8.5.4.

Pseudorandomness of Delegatable EVRFs. To capture the pseudorandomness property of delegatable EVRFs, it is clear that our definition should give the attacker the ability to call the delegation oracle Del. However, there are several subtleties in such a definition which we list below:

- Should we allow delegation queries from target SK_1 only to honestly generated keys (PK_2, SK_2) (for which the attacker does not know SK_2), or shall we allow the attacker $\mathcal{A}$ to specify such keys adversarially?

- For secretly-delegatable schemes, should the attacker $\mathcal{A}$ only have access to "OUT" oracle $\text{Del}(SK_1, \cdot, \cdot)$, or should we also give $\mathcal{A}$ the "IN" oracle $\text{Del}(\cdot, \cdot, SK_1)$ as well?[7] This corresponds to the attacker $\mathcal{A}$ tricking the user U to get some malicious EVRF from $\mathcal{A}$, only to force U to use its secret key SK_1 in a way that will help the attacker break some honest EVRF owned by U.

- What is the definitional security effect of studying one-time vs t-time vs unbounded-time delegatable schemes? As we will see, the security definitions *will not change syntactically* when restricted to t-time delegatable schemes. In particular, we will not explicitly limit the number of times the attacker can attempt to iteratively call the delegation oracle. However, since in such schemes the correctness is no longer required when delegating more than t times, it will be easier to make t-delegatable schemes secure when t is smaller.

- How to prevent the attacker from "trivial" attacks, where one can delegate ciphertext C_1 under PK_1 to C_2 under PK_2, and then "break" C_1 by asking an evaluation query on C_2? This is where the comparison procedure Same will be handy, as it allows us to precisely exclude

[7] Clearly, this is a moot issue for publicly-delegatable schemes.

all such ciphertexts C_2 which "originated" from C_1, while still allowing the attacker to try all other ciphertexts.[8]

Now, we can define the oracles. To adequately capture the discussed nuances, we define the following oracles to the attacker:

1. $\textsc{Reg}(1^\kappa)$: registration oracle. It maintains a global variable q, initially 0, counting the number of non-compromised users. A call to Reg: (a) increments q; (b) calls $(PK_q, SK_q) \leftarrow\!\!\$\ \textsc{Gen}(1^\kappa)$, (c) records this tuple (q, PK_q, SK_q) in a global table not accessible to the attacker; (d) returns (q, PK_q) to the attacker.

2. $\textsc{HProve}(i, C, x)$: honest evaluation oracle. Here $1 \leqslant i \leqslant q$ is an index, C is a ciphertext, and x in an input. The oracle returns $\textsc{Prove}(SK_i, C, x) = \textsc{Core}(SK_i, \textsc{Split}(PK_i, C), x)$.

3. $\textsc{HDel}(i, C, j)$: honest delegation oracle. Here $1 \leqslant i, j \leqslant q$ are two indices, and C is a ciphertext. The oracle returns $C' = \textsc{Del}(SK_i, C, SK_j)$ (or $\textsc{Del}(SK_i, C, PK_j)$ in the publicly-delegatable case).

4. $\textsc{OutDel}(i, C, SK/PK)$: "Out" delegation oracle. Here $1 \leqslant i \leqslant q$ is an index, C is a ciphertext, and PK or SK (depending on whether scheme is publicly-delegatable or not) is any public/secret key chosen by the attacker. The oracle returns $C' = \textsc{Del}(SK_i, C, SK/PK)$.

5. $\textsc{InDel}(SK, C, i)$: "In" delegation oracle. Here $1 \leqslant i \leqslant q$ is an index, and C is a ciphertext, and SK is any secret key chosen by the attacker. The oracle returns $C' = \textsc{Del}(SK, C, SK_i)$. Notice, this oracle is interesting only in the secretly-delegatable case.

Consequently, we can define three levels of pseudorandomness security for delegatable EVRFs.

Definition 8.17. An $EVRF = (\textsc{Gen}, \textsc{Encap}, \textsc{Comp}, \textsc{Split}, \textsc{Core}, \textsc{Post})$ is *delegatable* if there exists polynomial-time procedures Del and Same, such that:

[8] In Definition 8.21 we will give an even stronger (and optimal) legality condition; some of our schemes will satisfy even this stronger notion.

- $\textsc{Del}(SK_1, C_1, SK_2) = C_2$ for the (default) *secretly-delegatable* variant;
- $\textsc{Del}(SK_1, C_1, PK_2) = C_2$ for the *publicly-delegatable* variant.
- $\textsc{Same}(PK_1, C_1, PK_2, C_2) = \beta \in \{0, 1\}$.

Before we define the security properties, it is useful to define the following shorthand functions:

- $\textsc{Prove}(SK_i, C, x) = \textsc{Core}(SK_i, \textsc{Split}(PK_i, C), x)$
- $\mathsf{Eval}(SK, C, x) = \textsc{Post}(PK, \textsc{Prove}(SK, C, x), C, x)$

In addition to the standard EVRF properties of **Evaluation-Correctness** and **Uniqueness** , we require the following security properties from a delegatable EVRF:

1. **Delegation-Completeness**: for any valid (PK_1, SK_1), (PK_2, SK_2), and ciphertext C_1,

$$\textsc{Del}(SK_1, C_1, SK_2/PK_2) = C_2 \implies \textsc{Same}(PK_1, C_1, PK_2, C_2) = 1$$

2. **Delegation-Soundness**: for any valid (PK_1, SK_1), (PK_2, SK_2), and ciphertexts C_1, C_2

$$\textsc{Same}(PK_1, C_1, PK_2, C_2) = 1 \implies$$
$$\forall x\ \mathsf{Eval}(SK_1, C_1, x) = \mathsf{Eval}(SK_2, C_2, x)$$

   Moreover, if we have $PK_1 = PK_2$, then $C_1 = C_2$.

1. **Pseudorandomness under Core (\$-Core)**: for any legal PPT attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, where legality of $\mathcal{A}$ and appropriate delegation oracle(s) $\mathcal{O}$ are defined separately for each

notion:

$$\Pr\left[\, b = b' \;\middle|\; \begin{array}{c} (1, PK_1) \leftarrow\!\!\$\ \textsc{Reg}(1^\kappa); \\ (C_1, T_1) \leftarrow\!\!\$\ \textsc{Encap}(PK_1); \\ (x, st) \leftarrow\!\!\$\ \mathcal{A}_1^{\textsc{Reg},\textsc{HProve},\mathcal{O}}(PK_1, C_1); \\ y_0 = \textsc{Comp}(T_1, x);\ \ y_1 \leftarrow\!\!\$\ \{0,1\}^{|y_0|}; \\ b \leftarrow\!\!\$\ \{0,1\};\ \ b' \leftarrow\!\!\$\ \mathcal{A}_2^{\textsc{Reg},\textsc{HProve},\mathcal{O}}(y_b, st) \end{array} \right] \leq \frac{1}{2} + \mathrm{negl}(\kappa)$$

(a) **Basic-$-Core**: $\mathcal{A}$ has 1 delegation oracle $\mathcal{O} = \textsc{HDel}$.

Legality of $\mathcal{A}$: no call to $\textsc{HProve}(i, C', x)$ s.t.

$\textsc{Same}(PK_1, C_1, PK_i, C') = 1$.

(b) **Uni-$-Core**: $\mathcal{A}$ has 2 delegation oracles $\mathcal{O} = (\textsc{HDel}, \textsc{OutDel})$.

Legality of $\mathcal{A}$: no call to $\textsc{HProve}(i, C', x)$ or

$\textsc{OutDel}(i, C', *)$ $\textsc{stSame}(PK_1, C_1, PK_i, C') = 1$.

(c) **Bi-$-Core**: $\mathcal{A}$ has 3 delegation oracles

$\mathcal{O} = (\textsc{HDel}, \textsc{OutDel}, \textsc{InDel})$.

Legality of $\mathcal{A}$: same as that of **Uni-$-Core**.

Remark 14. **Delegation-Completeness** and **Delegation-Soundness** easily imply **Delegation-Correctness** which was advocated in Equation (8.2):

$$\textsc{Del}(SK_1, C_1, SK_2/PK_2) = C_2 \implies \forall x\ \mathsf{Eval}(SK_1, C_1, x) = \mathsf{Eval}(SK_2, C_2, x)$$

Remark 15. The legality condition on the attacker is necessary, as evaluating EVRF on the "same" ciphertext C' as the challenge ciphertext C_1 breaks pseudorandomness (by delegation-soundness). However, it leaves open the possibility for the attacker to find such equivalent ciphertext C' *without building some explicit "delegation path"* from the challenge ciphertext C_1. Indeed, in Definition 8.21 we will give an even stronger legality condition on $\mathcal{A}$, and some (but

not all) of our schemes will meet it. For applications, however, we do not envision this slight definitional gap to make any difference. Namely, the higher-level application will anyway need some mechanism to disallow any "trivial" attacks. We expect this mechanism will explicitly use our Same procedure, rather than keep track of the tree of "delegation paths" originating from C_1, which could quickly become unmanageable.

Remark 16. It is easy to observe the following implications:

$$\textbf{Bi-\$-Core} \implies \textbf{Uni-\$-Core} \implies \textbf{Basic-\$-Core} \implies \textbf{\$-Core}$$

Here, the last implication uses the fact that C_1 is the only ciphertext equivalent to C_1 under PK_1. Thus, bidirectional delegation security is the strongest of all the notions.

Remark 17. One could also consider EVRFs which are simultaneously threshold and delegatable. In this case, n_1 servers for the sender's EVRFs will communicate with n_2 servers for the receiver's EVRF to help convert a ciphertext C_1 for the sender EVRF into a corresponding ciphertext C_2 for the receiver EVRF. We leave this extension to future work.

8.5.2 Construction of Basic Delegatable EVRF

We now show that our original EVRF Construction 8.3 can be extended to make it basic-delegatable. The idea is to separate the role of the "handle" R hashed under H inside the Core procedure from the one used in the preprocessing. For technical reasons explained below, we will also hash the public key A when evaluating the EVRF. The construction is presented as Construction 8.5.

Observations. We notice that, since $R = D$ initially, the resulting EVRF before the delegation is the same as the one we defined in Section 8.3.4, except (a) we also include the public key A under the hash H during both Encap and Core; and (b) we perform the delegation check

Construction: Basic Delegatable EVRF

$\underline{\textsc{Gen}(1^\kappa)}$

Sample $a \in_r \mathbb{Z}_p^*$.
Compute $A = g^a \in G$.
return $SK = a$ and $PK = (g, A)$.

$\underline{\textsc{Encap}(PK)}$

Parse $PK = (g, A)$.
Sample $r \in_r \mathbb{Z}_p^*$.
Compute $R = D = g^r, S = A^r$.
return ciphertext $C = (A, R, D)$ and trapdoor $T = (A, R, S)$.

$\underline{\textsc{Comp}(T, x)}$

Parse $T = (A, R, S)$.
Compute $y = e(H(A, R, x), S)$.
return y.

$\underline{\textsc{Del}(SK_1, C_1, SK_2)}$

Parse $SK_1 = a_1, SK_2 = a_2, C_1 = (A, R, D_1)$.
if $e(A, R) \neq e(g^{a_1}, D_1)$ **then**
  return $\perp$.
else
  Compute $D_2 = D_1^{a_1/a_2}$ where $a_1/a_2 = a_1 \cdot (a_2)^{-1} \mod p$.
  return $C_2 = (A, R, D_2)$.

$\underline{\textsc{Split}(PK, C')}$

Parse $PK = (g, A)$, $C' = (A, R', D')$.
if $e(A', R') \neq e(A, D')$ **then**
  return $\perp$.
else
  return (A', R').

$\underline{\textsc{Core}(SK, C', x)}$

Parse $SK = a, C' = (A', R', D')$.
Compute partial output $z = H(A', R', x)^a$.
return z.

$\underline{\textsc{Post}(PK, z', C', x)}$

Parse $PK = (g, A)$, $C' = (A', R', D')$
if $e(z', g) \neq e(H(A', R', x), A)$ **then**
  return $\perp$.
else
  Compute full output $y' = e(z', D')$.
  return y'.

$\underline{\textsc{Same}(PK_1, C_1, PK_2, C_2)}$

Parse $PK_1 = (g, A_1), PK_2 = (g, A_2), C_1 = (A, R, D_1), C_2 = (A', R', D_2)$.
if $(A, R) \neq (A', R')$ or $e(A_1, D_1) \neq e(A_2, D_2)$ **then**
  return $\perp$.

Construction 8.5: Basic Delegatable $\mathsf{DEVRF}_1 = (\textsc{Gen}, \textsc{Encap}, \textsc{Comp}, \textsc{Split}, \textsc{Core}, \textsc{Post}, \textsc{Del}, \textsc{Same})$.

$e(A', R') \stackrel{?}{=} e(A, D')$ in the split procedure $\textsc{Split}$, which is trivially true initially, as $A' = A$ and $R' = D' = R$. Thus, **Evaluation-Correctness** trivially holds, as before. For the same reason, **Uniqueness** trivially holds as well.

The importance of change (a) comes from the fact that challenge ciphertext $C = (A, R, D)$ no longer includes only the value R, even though the value R would be all that is needed to actually evaluate our EVRF, had we not included A under the hash H. In particular, the attacker $\mathcal{A}$ given challenge $C = (A, R, R)$, can easily produce $C' \neq C$ by setting $C' = (A^2, R, R^2)$. C' passes the delegation check $e(A^2, R) = e(A, R^2)$, but clearly produces the same partial output $z = H(R, x)^a$ as the challenge ciphertext, trivially breaking the **$-Core** property. Instead, by also hashing the public key, the oracle call $\textsc{Prove}(C', x)$ would return $z' = H(A^2, R, x)^a$, which is now unrelated to $z = H(A, R, x)^a$, foiling the trivial attack.

The importance of change (b) comes from ensuring that a valid ciphertext (A', R', D') determines the value D' *information-theoretically* from the values (A', R') (and the public key A),

because the condition $e(A', R') = e(A, D')$ uniquely determines D'. Thus, it is OK that the Core procedure only passes the values (A', R') under the random oracle H.

Delegation.

- **Delegation-Completeness**: Notice that valid delegation of (A, R, D_1) outputs (A', R', D_2), where $(A', R') = (A, R)$ and $D_2 = D_1^{\frac{a_1}{a_2}}$, which implies that

$$e(A_2, D_2) = e(g^{a_2}, D_1^{\frac{a_1}{a_2}}) = e(g^{a_1}, D_1) = e(A_1, D_1)$$

  which means $\textsc{Same}(A_1, (A, R, D_1), A_2, (A', R', D_2)) = 1$ indeed.

- **Delegation-Soundness**: Given $C_1 = (A, R, D_1)$ and $C_2 = (A', R', D_2)$ satisfying $(A', R') = (A, R)$ and $e(A_1, D_1) = e(A_2, D_2)$, we can see that the delegation checks $e(A, R) \stackrel{?}{=} e(A_1, D_1)$ and $e(A', R') \stackrel{?}{=} e(A_2, D_2)$ are either both false or true simultaneously. Moreover, by writing $A_1 = A_2^{\frac{a_1}{a_2}}$, the second equation implies that $D_2 = D_1^{\frac{a_1}{a_2}}$. In particular, if $A_1 = A_2$, we have $C_1 = C_2$; and, in general, when $(A', R') = (A, R)$ and $D_2 = D_1^{\frac{a_1}{a_2}}$, for any x, we know: $\mathsf{Eval}(a_2, (A, R, D_2), x) = e(H(A, R, x)^{a_2}, D_2)$.

  However, that can be rewritten as

$$e(H(A, R, x)^{a_2}, D_1^{\frac{a_1}{a_2}}) = e(H(A, R, x)^{a_2}, D_1^{\frac{a_1}{a_2}}) = e(H(A, R, x)^{a_1}, D_1)$$

  which concludes the proof.

We reiterate that though our delegation is secretly-delegatable, as D_2 depends on a_2, in practice the owner Alice of a_1 will simply send the trapdoor value $T_1 = D_1^{a_1}$ to the owner Bob of a_2 over secure channel (say, encrypted under a separate public key), and Bob can then compute $D_2 = T_1^{1/a_2}$. In particular, this does not leak any extra information beyond (D_2, a_2) to Bob, as $T_1 = D_2^{a_2}$ is efficiently computable from D_2 and a_2. Also, the delegation check does not require

any of the secret keys. Despite that, it ensures that only properly delegated ciphertexts can be securely re-delegated again.

Theorem 8.18. *The basic delegatable EVRF, given in Construction 8.5, satisfies the* ***Basic-$-Core*** *property under the* BDDH *assumption in the random oracle model.*

Proof. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a PPT attacker against the **Basic-$-Core** property of DEVRF_1, having advantage ϵ. $\mathcal{A}$ is given the public key $A_1 = g^{a_1}$ (for unknown a_1), challenge ciphertext $C_1 = (A_1, R_1, R_1)$ (where $R_1 = g^r$ for unknown r), and has oracle access to 4 oracles: random oracle H, registration oracle Reg, honest evaluation oracle HProve, and honest delegation oracle HDel.

Legality of $\mathcal{A}$. Let u be the polynomial upper bound on the number of overall honest public/secret keys (including the challenge) used by A (meaning $\mathcal{A}$ made at most $u-1$ registration queries). Denote corresponding key pairs $\{(a_i, A_i = g^{a_i})\}_{i=1}^{u}$, so that $a_1 = a$. Define $D_i = R_1^{a_1/a_i}$ and $C_i = (A_1, R_1, D_i)$, for $i = 1 \ldots u$. The legality condition on $\mathcal{A}$ states what $\mathcal{A}$ is not able to call $\text{HProve}(i, C'_i, x)$, where x is the challenge input produced by $\mathcal{A}_1$, for any $C'_i = (A'_i, R'_i, D'_i)$ where $\text{Same}(A_1, C_1, A_i, C'_i) = 1$. This means $(A'_i, R'_i) = (A_1, R_1)$ and $e(A_1, R_1) = e(A_i, D'_i)$. It follows that $D'_i = D_i = R_1^{a_1/a_i}$ and overall $C'_i = C_i$. Hence, in our reduction to BDDH we can assume that $\mathcal{A}$ never calls $\text{HProve}(i, (A_1, R_1, D_i), x)$, for any $i \in [u]$. Moreover, for any $D'_i \neq D_i$, a call $\text{HProve}(i, (A_1, R_1, D'_i), x)$ returns $\perp$, since the value D'_i does not pass the delegation check: $e(A_1, R_1) = e(A_i, D_i) \neq e(A_i, D'_i)$. To sum up this discussion, without loss of generality in our reduction to BDDH, we can assume that

Condition (*): *$\mathcal{A}$ never calls $\text{HProve}(i, (A_1, R_1, D), x)$, for any $i \in [u]$ and any D*

Namely, these calls either immediately return $\perp$ (if $D \neq D_i$, and this can be checked by $\mathcal{A}$), or disallowed anyway.

Our Reduction. Our reduction, which uses $\mathcal{A}$ to build a BDDH attacker $\mathcal{B}(g, A, B, R, g_1)$, will proceed very similarly to the proof of Theorem 8.12, but with several small modifications:

1. **Initialization.** $\mathcal{B}$ will set the challenge key $A_1 = A$ (implicitly keeping the secret key $a_1 = a$ unknown), and the challenge ciphertext is $C_1 = (A, R, R)$, so we set $R_1 = D_1 = R$.

2. **Registration Oracle Reg.** $\mathcal{A}$ can set some polynomial number of honest key pairs $\{(A_i, a_i)\}_{i=2}^{u}$, where $A_i = g^{a_i}$. In our reduction, we set each such $A_i = A^{\alpha_i}$, for random $\alpha_i \in \mathbb{Z}_p$ chosen by $\mathcal{B}$. Since the α_i's are random, these keys are correctly distributed. For ease of notation we set $\alpha_1 = 1$.

   (Note, even though $\mathcal{B}$ does not know any of the secret keys $a_i = \alpha_i a \bmod p$, $\mathcal{B}$ can compute the ratio $a_i/a_j = \alpha_i/\alpha_j$.)

3. **Honest Delegation Oracle HDel.** When $\mathcal{A}$ calls

   HDel$(i, (A', R', D'), j)$, $\mathcal{B}$ can perform the delegation check $e(A', R') \stackrel{?}{=} e(A_i, D')$ itself, and then can compute $(D')^{a_i/a_j} = (D')^{\alpha_i/\alpha_j}$ without the knowledge of any of the a_i's.

4. **Hash Queries to H:** previously, the oracle H used by the Core procedure was only evaluated on the values (R', x') given as input, but now we evaluate H on the tuple (A', R', x'). Indeed, we already observed a simple attack showing that the construction is insecure if we only hash the value (R', x'). Nevertheless, our simulation of H remains unchanged, modulo now accepting (A', R', x') as input to H, rather than only (R', x'). Namely, every such fresh evaluation $H(A', R', x')$ chooses values $(\beta_i, coin_i)$, where i is the query index, as in the proof of Theorem 8.12, and still sets

   $$H(A', R', x') = g^{\beta_i} \text{ if } coin_i = 0, \text{ and } H(A', R', x') = B^{\beta_i} \text{ if } coin_i = 1.$$

5. **Honest Prove Oracle HProve.** The oracle

   HProve$(i, (A', R', D'), x')$ first checks that $e(A', R') = e(A_i, D')$, and then proceeds as before in the proof of Theorem 8.12. In particular, it aborts if the value of $coin_j$ corresponding to the oracle query $H(A', R', x')$ is 1, and otherwise (meaning $H(A', R', x') = g^{\beta'}$ for some random β') returns $z' = A_i^{\beta'}$, as before. This is correct since $z' = A_i^{\beta'} = g^{a_i \beta'} = H(A', R', x')^{a_i}$

(for unknown a_i).

6. **Challenge value y.** This is returned as before, by evaluating $H(A_1, R_1, x)$ and aborting if the value of $coin_j$ corresponding to the oracle query $H(A_1, R_1, x)$ is 0 (i.e., $H(A_1, R_1, x) = B^\beta$ for some random β if we do not abort). The challenge value y is then set to g_1^β, as before.

7. **Finishing.** If the simulation succeeds until the end, $\mathcal{B}$ outputs the same b' as $\mathcal{A}$.

Analysis of Reduction. We claim that the proof of the security of this reduction can go exactly as in Theorem 8.12. The only subtle point comes in the proof of Claim 8.13, where we argue that the values $(\beta', coin')$ sampled during the emulation of the evaluation oracle HProve$(i, (A', R', D'))$ are chosen *independently* from the value $(\beta, coin)$ used to emulate the challenge query (A_1, R_1, x). This is indeed essential since we want all former coins to the 0, and the latter coin to be 1.

Fortunately, it immediately follows from Condition (*), as all evaluation queries on challenge x, must use $(A', R') \neq (A_1, R_1)$. Thus, we never have a conflict, and the proof of Claim 8.13 holds. □

Delegation Attack on Stronger Legality. We briefly mentioned in Section 8.5.1 that one could require a stronger legality condition to say that the only way to distinguish the evaluation of C on x from random is to honestly delegate C to some honest user (possibly iteratively), getting ciphertext C', and then ask this user to evaluate EVRF on x.

Here we show that our construction does not satisfy this notion. Consider challenge ciphertext $C_1 = (A_1, R_1, R_1)$ under public key A_1. Construct $C_1' = (A_1, R_1^2, R_1^2)$. C_1' will satisfy the delegation check, so we could ask to delegate C' to public key A_2. We get $C_2' = (A_1, R_1^2, (R_1^2)^{\frac{a_1}{a_2}}) = (A_1, R_1^2, (R_1^{\frac{a_1}{a_2}})^2)$. By taking square roots from the last two components, we get $C_2 = (A_1, R_1, R_1^{\frac{a_1}{a_2}})$. Notice, Same$(A_1, C_1, A_2, C_2) = 1$ is true, so our original definition does *not* permit the attacker to evaluate HProve$(2, C_2, x)$ (which clearly breaks the scheme). However, since we obtained C_2

Construction: Delegatable EVRF

$\underline{\textsc{Gen}(1^\kappa)}$

Sample $a \in_r \mathbb{Z}_p^*$.
Compute $A = g^a \in G$.
return $SK = a$ and $PK = (g, A)$.

$\underline{\textsc{Encap}(PK)}$

Parse $PK = (g, A)$.
Sample $r \in_r \mathbb{Z}_p^*$.
Compute $R = D = g^r, S = A^r, \sigma = H'(A, R)^r$.
return ciphertext $C = (A, R, D, \sigma)$ and trapdoor $T = (A, R, S)$.

$\underline{\textsc{Comp}(T, x)}$

Parse $T = (A, R, S)$.
Compute $y = e(H(A, R, x), S)$.
return y.

$\underline{\textsc{Del}(SK_1, C_1, SK_2)}$

Parse $SK_1 = a_1, SK_2 = a_2, C_1 = (A, R, D_1, \sigma)$.
if $e(A, R) \neq e(g^{a_1}, D_1)$ or $e(H'(A, R), R) \neq e(\sigma, g)$ **then**
  return $\perp$.
else
  Compute $D_2 = D_1^{a_1/a_2}$ where $a_1/a_2 = a_1 \cdot (a_2)^{-1} \mod p$.
  return $C_2 = (A, R, D_2)$.

$\underline{\textsc{Split}(PK, C')}$

Parse $PK = (g, A)$, $C' = (A', R', D', \sigma')$.
if $e(A', R') \neq e(A, D')$ or $e(H'(A', R'), R') \neq e(\sigma', g)$ **then**
  return $\perp$.
else
  return (A', R').

$\underline{\textsc{Core}(SK, C', x)}$

Parse $SK = a, C' = (A', R', D', \sigma')$.
Compute partial output $z = H(A', R', x)^a$.
return z.

$\underline{\textsc{Post}(PK, z', C', x)}$

Parse $PK = (g, A)$, $C' = (A', R', D', \sigma')$
if $e(z', g) \neq e(H(A', R', x), A)$ **then**
  return $\perp$.
else
  Compute full output $y' = e(z', D')$.
  return y'.

$\underline{\textsc{Same}(PK_1, C_1, PK_2, C_2)}$

Parse $PK_1 = (g, A_1), PK_2 = (g, A_2), C_1 = (A, R, D_1, \sigma), C_2 = (A', R', D_2, \sigma')$.
if $(A, R, \sigma) \neq (A', R', \sigma')$ or $e(A_1, D_1) \neq e(A_2, D_2)$ **then**
  return $\perp$.

Construction 8.6: $\mathsf{DEVRF}_2 = (\textsc{Gen}, \textsc{Encap}, \textsc{Comp}, \textsc{Split}, \textsc{Core}, \textsc{Post}, \textsc{Del}, \textsc{Same})$.

without asking the delegate C_1 itself (instead, we asked a different ciphertext C_1'), the stronger notion would have allowed the attacker to call $\textsc{HProve}(2, C_2, x)$ and break the scheme.

8.5.3 Construction of Uni- and Bidirectional Delegatable EVRF

Next, we extend the construction from the previous EVRF construction to also handle delegation *to* (and, under a stronger assumption, *from*) potentially untrusted parties. The idea is to add a "BLS signature" [BLS01] σ in the $\textsc{Encap}$ procedure which will prove that the initial ciphertext was "well-formed". This makes it hard for the attacker to maul a valid initial ciphertext C into a related ciphertext C', whose delegation might compromise the security of C. The public verifiability of the signature σ will also make it easy to add a "signature check" to the "delegation check" we already used in our scheme, to ensure that the appropriate pseudorandomness property is not compromised. This is presented as Construction 8.6.

Security Analysis. Since DEVRF_2 is essentially the same as DEVRF_1, its correctness follows the same argument. In particular, we notice that the original signature σ indeed satisfies our signature check:

$$e(H'(A,R),R) = e(H'(A,R),g^r) = e(H'(A,R)^r,g) = e(\sigma,g)$$

Similar to the delegation check, the signature check, $e(H'(A',R'),R') \stackrel{?}{=} e(\sigma',g)$, is important to ensure that the value σ' is information-theoretically determined from the value (A',R'), so it is fine to not include σ under H.

Also, since the delegation procedure Del simply copies the values A, R and σ, and only modifies the value D_1, the **Delegation-Completeness** and **Delegation-Soundness** of DEVRF_2 holds as it did for DEVRF_1, since the signature check is not affected by changing D_1 to $D_2 = D_1^{\frac{a_1}{a_2}}$. In particular, similar to the delegation checks, both signature checks are either simultaneously true or false.

Now, we can show how the addition of the "BLS signature" σ and the new signature check allow us to prove the following theorem:

Theorem 8.19. *The delegatable EVRF given in Construction 8.6 satisfies the* ***Uni-$-Core*** *property under the* BDDH *assumption in the random oracle model.*

Proof. let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a PPT attacker against the **Uni-$-Core** property of DEVRF_2, having advantage ϵ. $\mathcal{A}$ is given the public key $A_1 = g^{a_1}$ (for unknown a_1), challenge ciphertext $C_1 = (A_1, R_1, R_1, \sigma_1)$ (where $R_1 = g^r$ for unknown r and $\sigma_1 = H'(A_1, R_1)^{r_1}$), and has oracle access to 6 oracles: random oracles H, H', registration oracle Reg, honest evaluation oracle HProve, honest delegation oracle HDel, and "OUT" delegation oracle OutDel. Note, oracles H' and OutDel are new compared to the the proof of Theorem 8.18. Still, our proof will mimic almost exact the proof of Theorem 8.18, so we will use the same notation as in that proof, and only mention the key differences in our reduction.

LEGALITY OF $\mathcal{A}$: As before, we denote $u-1$ honest keys by $\{(a_i, A_i = g^{a_i})\}_{i=2}^{u}$, and define $D_i = R_1^{a_1/a_i}$, $C_i = (A_1, R_1, D_i, \sigma_1)$, for $i = 1 \ldots u$. The legality condition on $\mathcal{A}$ states what $\mathcal{A}$ is not able to call HPROVE(i, C_i', x) or OUTDEL$(i, C_i', *)$, where x is the challenge input produced by $\mathcal{A}_1$, for any $C_i' = (A_i', R_i', D_i', \sigma')$ where SAME$(A_1, C_1, A_i, C_i') = 1$. This means $(A_i', R_i', \sigma') = (A_1, R_1, \sigma_1)$ and $e(A_1, R_1) = e(A_i, D_i')$. But this means $D_i' = D_i = R_1^{a_1/a_i}$ and overall $C_i' = C_i$. Moreover, as in the the proof of Theorem 8.18, we can assume without loss of generality that $\mathcal{A}$ will not use ciphertext $C_i' = (A_1, R_1, D', \sigma')$ for any $(D', \sigma') \neq (D_i, \sigma_1)$, as those will not pass the delegation or the signature check of either HPROVE or OUTDEL. Hence, similar to the proof of Theorem 8.18, we can we can assume that

Condition (**): *$\mathcal{A}$ never calls HPROVE$(i, (A_1, R_1, D, \sigma), x)$ or OUTDEL$(i, (A_1, R_1, D, \sigma), *)$, for any $i \in [u]$ and any D, σ.*

OUR REDUCTION. Our reduction, which uses $\mathcal{A}$ to build a BDDH attacker $\mathcal{B}(g, A, B, R, g_1)$, will proceed very similarly to the proof of Theorem 8.18, but with several small modifications:

1. **Hash Queries to H':** $\mathcal{B}$ will maintain a table T' containing entries of the form (A', R', γ), where $A', B' \in G$ and $\gamma \in \mathbb{Z}_p$. To create the first such entry in T', $\mathcal{B}$ chooses a random value $\gamma_1 \in_r \mathbb{Z}_p$, and stores the tuple (A, R, γ_1), where A and R come from $\mathcal{B}$'s challenge. After T' is initialized, as above, all future queries $H'(A', R')$ are answered as follows.

   If $(A', R') = (A, R)$, respond with g^{γ_1}, meaning that we set $H'(A, R) = g^{\gamma_1}$.

   Otherwise, check if T' has an entry of the form (A', R', γ'). If not, pick a random $\gamma' \leftarrow\!\!\$_r \mathbb{Z}_p$ and add the tuple (A', R', γ') to T'. In either case, return $H'(A', R') = (A')^{\gamma'}$.

2. **Initialization.** $\mathcal{B}$ will set the challenge key $A_1 = A$ (implicitly keeping the secret key $a_1 = a$ unknown), and the challenge ciphertext is $C_1 = (A_1, R_1, D_1, \sigma_1)$, where $R_1 = D_1 = R$, and signature $\sigma_1 = R^{\gamma_1}$.

   (Note, since $H'(A_1, R_1) = g^{\gamma_1}$, we have that $\sigma_1 = R^{\gamma_1} = (g^r)^{\gamma_1} = (g^{\gamma_1})^r = H'(A_1, R_1)^r$.)

3. **Registration Oracle Reg.** Same as the proof of Theorem 8.18. In particular, each public key $A_i = A^{\alpha_i}$, for random $\alpha_i \in \mathbb{Z}_p$ chosen by $\mathcal{B}$, and we set $\alpha_1 = 1$. This means that even though $\mathcal{B}$ does not know any of the secret keys $a_i = \alpha_i a \bmod p$, $\mathcal{B}$ can compute the ratio $a_i/a_j = \alpha_i/\alpha_j$.

4. **Honest Delegation Oracle HDel.** When $\mathcal{A}$ calls

   $\text{HDel}(i, (A', R', D', \sigma'), j)$, $\mathcal{B}$ can perform the delegation check $e(A', R') \stackrel{?}{=} e(A_i, D')$ and the signature check

   $e(H'(A', R'), R') \stackrel{?}{=} e(\sigma', g)$ himself (by evaluating H' according to the standard strategy above). Then, $\mathcal{B}$ then can compute $(D')^{a_i/a_j} = (D')^{\alpha_i/\alpha_j}$ without the knowledge of any of the a_i's.

5. **"OUT" Delegation Oracle OutDel.** When $\mathcal{A}$ calls

   $\text{OutDel}(i, (A', R', D', \sigma'), a^*)$, for any secret key a^*, $\mathcal{B}$ can perform the delegation check $e(A', R') \stackrel{?}{=} e(A_i, D')$ and the signature check $e(H'(A', R'), R') \stackrel{?}{=} e(\sigma', g)$ himself (by evaluating H' according to the standard strategy above). Moreover, from Condition (**) we know that $(A', R') \neq (A_1, R_1)$. Thus, when we set the value $H'(A', R')$ in our simulation of H', we set it to $H'(A', R') = (A')^{\gamma'}$ for some random γ' known to $\mathcal{B}$. $\mathcal{B}$ will combine this γ' with the values of signature σ' and secret key a^* given by $\mathcal{A}$, and return

$$\text{OutDel}(i, (A', R', D', \sigma'), a^*) := (\sigma')^{(a^*\gamma')^{-1} \bmod p}$$

   To check that this value indeed equals to $(D')^{a_i/a^*}$, it suffices to prove that $(D')^{a_i} = (\sigma')^{1/\gamma'}$. To see that, our signature check implies that

$$e(H'(A', R'), R') = e(\sigma', g) \implies e((A')^{\gamma'}, R') = e(\sigma', g)$$

This further implies that:

$$e(A', R') = e((\sigma')^{1/\gamma'}, g)$$

Finally, combined with the delegation check $e(A', R') = e(A_i, D')$ and $A_i = g^{a_i}$, we get that

$$e((\sigma')^{1/\gamma'}, g) = e(A', R') = e(A_i, D') = e((D')^{a_i}, g)$$

which implies that

$$(D')^{a_i} = (\sigma')^{1/\gamma'}$$

6. **Hash Queries to H:** These are identical to the proof of Theorem 8.18.

7. **Honest Evaluation Oracle HProve.** The oracle HProve$(i, (A', R', D', \sigma'), x')$ first checks that $e(A', R') = e(A_i, D')$ and $e(H'(A', R'), R') = e(\sigma', g)$ (by evaluating H' according to the standard strategy above). After that, it proceeds exactly like the proof of Theorem 8.18.

8. **Challenge value y.** This is returned as before, by evaluating $H(A_1, R_1, x)$ and aborting if the value of $coin_j$ corresponding to the oracle query $H(A_1, R_1, x)$ is 0 (i.e., $H(A_1, R_1, x) = B^{\beta}$ for some random β if we do not abort). The challenge value y is then set to g_1^{β}, as before.

9. **Finishing.** If the simulation succeeds until the end, $\mathcal{B}$ outputs the same b' as $\mathcal{A}$.

Analysis of Reduction. The analysis of the reduction then goes exactly as in Theorem 8.18 (which in turn is based on that in Theorem 8.12). As before, the only subtle point comes in the proof of Claim 8.13, where we argue that the emulation of H during the evaluation queries Prove does not conflict with its emulation for the challenge query $H(A_1, R_1, x)$.

Fortunately, this immediately follows from our Condition (**) above, as all evaluation queries on challenge x must use $(A', R') \neq (A_1, R_1)$. Thus, we never have a conflict, and the proof of Claim 8.13 holds. □

Finally, we also show that the same construction also satisfies the strongest *bidirectional-delegation* security, but now under a much stronger iBDDH assumption. In fact, for this result, we will even show a *stronger legality* condition mentioned earlier: the only way to break DEVRF_2 is to trivially delegate it "out" to the attacker, or delegate it to the honest user, and then ask the user to evaluate on challenge x. We define this formally in proof of the following theorem:

Theorem 8.20. *The delegatable EVRF given in Construction 8.6 satisfies the **Bi-$-Core** property under the interactive* iBDDH *assumption in the random oracle model. It satisfies the strongest possible legality condition for the attacker (see Definition 8.21).*

Proof. Since the construction we analyze is identical to that of Theorem 8.19, our proof will be almost identical as well. Now, however, we also need to show how to simulate the "IN" delegation oracle $\textsc{InDel}(a^*, (A', R', D', \sigma'), i)$. This appears hard, since the values a^*, A', R', D', σ' are adversarial, while the answer expected by the attacker should be equal (assuming the ciphertext is well formed) to $(D')^{a^*/a_i}$. Recall also that in our simulation we set $a_i = \alpha_i a$, where α_i were known by $\mathcal{B}$, but a was unknown. Thus, our reduction $\mathcal{B}$ must be able to compute the value

$$D = (D')^{a^*/\alpha_i a} = ((D')^{a^*/\alpha_i})^{1/a}$$

Fortunately, we are now reducing from the inversion-oracle BDDH (iBDDH) assumption, given in Section 8.1.2.3. Namely, in addition to its standard BDDH inputs, $\mathcal{B}$ also has oracle access to $\mathcal{O}_a(h) = h^{1/a}$. With this access, the simulation of
$\textsc{InDel}(a^*, (A', R', D', \sigma'), i)$ becomes trivial: $\mathcal{B}$ simply returns
$\mathcal{O}_a((D')^{a^*/\alpha_i}) = (D')^{a^*/a_i}$. This completes the first part of our proof. □

Stronger Legality Condition. We now show that under the iBDDH assumption we can actually strengthen the legality condition of $\mathcal{A}$ to be optimal; informally, only trivially delegated/evaluated ciphertexts can be broken by $\mathcal{A}$. We formalize this below.

Definition 8.21. If $\mathcal{A}$ makes a call $C' = \text{HDEL}(i, C, j)$, we say $\mathcal{A}$ creates a *delegation edge* from (i, C) to (j, C'), denoted $(i, C) \rightarrow (j, C')$. A sequence of delegation edges $(i_1, C_1) \rightarrow (i_2, C_2) \rightarrow \ldots \rightarrow (i_t, C_t)$ defines a *delegation path* from (i_1, C_1) to (i_t, C_t), denoted $(i_1, C_1) \rightsquigarrow (i_t, C_t)$.

We now define a stronger legality condition on $\mathcal{A}$ (focusing on the bidirectional splittable case):

- Given challenge ciphertext C_1 on user 1, $\mathcal{A}$ produced no delegation path $(1, C_1) \rightsquigarrow (i, C')$, followed by either a call to $\text{HPROVE}(i, C', x)$, or a call to $\text{OUTDEL}(i, C', *)$, where x is the challenge input returned by $\mathcal{A}_1$.

Thus, the only "prohibited" ciphertexts C' must have been explicitly obtained by the attacker. In contrast, our original Definition 8.17 only prevented C' directly satisfying $\text{SAME}(PK_1, C_1, PK_i, C') = 1$. While reasonable in many situations, the "gap" between the two notions involves an attacker who managed to find a ciphertext C' satisfying $\text{SAME}(PK_1, C_1, PK_i, C') = 1$ *without* creating a delegation path $(1, C_1) \rightsquigarrow (i, C')$.

Indeed, we saw that our unidirectional construction DEVRF_2 was trivially insecure wrt the stronger (and, clearly, optimal) legality condition on $\mathcal{A}$. However, when looking at our current construction DEVRF_2, we see that the existence of the signature σ in the ciphertext appears to foil the trivial attack from Section 8.5.2. In fact, we show that is not luck, but the construction is actually secure w.r.t. to this stronger legality condition. Unfortunately, for this, we must rely on the stronger iBDDH assumption (even for basic or unidirectional security, so we might as well get the optimal bidirectional security).

STRONGER LEGALITY FOR DEVRF_2. Consider any attacker $\mathcal{A}$ against **Bi-$-Core** security of DEVRF_2 which satisfies the stronger legality condition, and has advantage ϵ. Recalling all the notation we used in the proof of Theorem 8.19, the challenge ciphertext $C_1 = (A_1, R_1, D_1, \sigma)$, where $D_1 = R_1$, and u "special" ciphertexts which we cannot fully handle (see below) in our reduction are $C_i = (A_1, R_1, D_i, \sigma)$, where $D_i = R_1^{a_i/a_1}$. Indeed, these are the only ciphertexts satisfying

$$\text{SAME}(A_1, C_1, A_i, C_i) = 1$$

Define the event G to denote the "gap" between the two legality conditions of $\mathcal{A}$; namely,

- Let G be the event that $\mathcal{A}$ made a call to $\textsc{HProve}(i, C_i, x)$ or $\textsc{OutDel}(i, C_i, *)$ for some $i \in [u]$, where x is the challenge input produced by $\mathcal{A}_1$, but did not create a delegation path $(1, C_1) \rightsquigarrow (i, C_i)$.

Expanding the definition of $\mathcal{A}$'s advantage ϵ, we get that

$$\begin{aligned} \frac{1}{2} + \epsilon &= \Pr[[]](b' = b) \wedge G] + \Pr[[]](b' = b) \wedge \neg G] \\ &\leqslant \Pr[[]G] + \Pr[[]](b' = b) \wedge \neg G] \end{aligned}$$

Thus, to prove that $\epsilon = \text{negl}(\kappa)$ under the iBDDH assumption, it suffices to prove the following two Lemmas:

Lemma 8.22. *Under the* iBDDH *assumption,* $\Pr[[]G] = \text{negl}(\kappa)$.

Lemma 8.23. *Under the* iBDDH *assumption,* $\Pr[[]](b' = b) \wedge \neg G] \leqslant \frac{1}{2} + \text{negl}(\kappa)$.

Lemma 8.23 is exactly the proof of security of DEVRF_2 we just finished at the beginning of this section above, as this corresponds to the run of $\mathcal{A}$ satisfying the original legality condition. Thus, to show the security of DEVRF_2 under the stronger legality condition, we only need to prove Lemma 8.22.

Proof of Lemma 8.22. Let use call a query Q of $\mathcal{A}$ *violating* if it triggers the event G, meaning that for some index $i \in [u]$ this query is:

- either $\textsc{OutDel}(i, C_i, *)$, where there is no delegation path $(1, C_1) \rightsquigarrow (i, C_i)$ so far;
- or $\textsc{HProve}(i, C_i, x)$, where there is no delegation path $(1, C_1) \rightsquigarrow (i, C_i)$ so far.

Let us also consider a dynamic "delegation graph" $M = (V, E)$ consisting of all the delegation edges of the form $(i, C_i) \rightarrow (j, C_j)$. (Namely, we only look at u special ciphertexts C_i and ignore

the rest.) The edge set E of this graph starts empty, but eventually could grow when $\mathcal{A}$ makes honest delegation query $\text{HDEL}(i, C_i, j)$ (which returns C_j). Moreover, without loss of generality, we assume M is acyclic, as $\mathcal{A}$ get no information by completing the cycle in this graph (i.e., we can simply remove all such edges creating cycles, as $\mathcal{A}$ already knows the answer).

Let Q be the *first* violating query of $\mathcal{A}$, and $j \in [u]$ be the corresponding index of this query. By assumption that G is triggered, Q and j are well defined. Moreover, if E is the current edge set of the delegation graph M, we know that there is no delegation path $(1, C_1) \rightsquigarrow (j, C_j)$ in E. However, there could potentially be incoming edges from some $(i, C_i) \rightarrow (j, C_j)$ in E, as long as there is no delegation path $(1, C_1) \rightsquigarrow (i, C_i)$. Going backward from (j, C_j), though, since we know that M is acyclic, we must reach some "source" (i, C_i) where $i > 1$, which has no incoming edges at all (and, hence, no path from $(1, C_1)$ still). Let (i, C_i) be such "source node", which could be the original (j, C_j) in the special case where no delegation edges entered (j, C_j). In either case, however, we know that the query Q^* corresponding to the first time ciphertext C_i appeared in either $\text{HDEL}(i, C_i, *)$, $\text{HPROVE}(i, C_i, x)$ or $\text{OUTDEL}(i, C_i, *)$ had the property that no delegation path $(1, C_1) \rightsquigarrow (i, C_i)$ existed.

In other words, there must exist a query Q^* of $\mathcal{A}$ and an index $i > 1$ such that:

- Q^* is either $\text{HDEL}(i, C_i, *)$, $\text{HPROVE}(i, C_i, x)$ or $\text{OUTDEL}(i, C_i, *)$, and no prior calls of this form where made so far (i.e., ciphertext C_i was not "declared" by $\mathcal{A}$ before Q^*).
- At the time Q^* is made, there is no delegation path $(1, C_1) \rightsquigarrow (i, C_i)$.

Let us say that such query Q^* is *i-incriminating*. Note, from our definition, each index $i > 1$ could have either zero or one i-incriminating query. To summarize,

$$\textit{event } G \qquad \Longrightarrow \qquad \textit{there exists } 1 < i \leqslant u \textit{ having (unique) } i\textit{-incriminating query } Q^*.$$

USING INCRIMINATING QUERY. Recall, in our proof of **Bi-$-Core** security of DEVRF_2 under the original legality condition we constructed an attacker $\mathcal{B}^{\mathcal{O}_a(\cdot)}(g, A, B, R, g_1)$ for iBDDH which could

simulate all queries of $\mathcal{A}$ except $\text{HPROVE}(j, C_j, x)$ and $\text{OUTDEL}(j, C_j, *)$, which were prohibited under the original legality condition of $\mathcal{A}$.

We will now construct a different iBDDH attacker $\mathcal{B}^{O_a(\cdot)}(g, A, B, R, g_1)$ which will instead use the fact that $\mathcal{A}$ must make an i-incriminating query for some $1 < i \leqslant u$. $\mathcal{B}$ will pick a random index $i \in [2, \ldots u]$, hoping that this is the index corresponding to the i-incriminating query Q^*. Assuming this guess is correct, we know that Q^* appears before (or exactly at) the first violating query of $\mathcal{A}$, which means that we could have used (but we won't!) the original reduction $\mathcal{B}$ to simulate all the queries of $\mathcal{A}$ before Q^*, as none of these queries will have the form $\text{HPROVE}(j, C_j, x)$ or $\text{OUTDEL}(j, C_j, *)$.

More precisely, $\mathcal{B}$ will proceed nearly identically to the original reduction $\mathcal{B}$, but with the following modifications.

1. **Same Simulation.** Initialization of $\mathcal{A}$, challenge ciphertext C_1, challenge output y, and oracles H, H', HPROVE, and OUTDEL are identical, except when OUTDEL or HPROVE query is i-incriminating, as explained below.

2. **Registration Oracle REG.** Previous attacker $\mathcal{B}$ set all value $A_j = A_1^{\alpha_j}$ for $2 \leqslant j \leqslant u$, implicitly setting $a_j = \alpha_j a$. $\mathcal{B}$ will do the same for all $j \neq i$. However, for the i-th secret key $\mathcal{B}$ will chose a random key $a_i \in_r \mathbb{Z}_p$ and honestly set $A_i = g^{a_i}$. In other words, $\mathcal{B}$ will actually know the i-th secret key.

3. **i-Incriminating Query Q^*.** Recall, such queries are
   $\text{HDEL}(i, C_i, *)$, $\text{HPROVE}(i, C_i, x)$ or $\text{OUTDEL}(i, C_i, *)$. Notice, $\mathcal{B}$ can test if a ciphertext $C = (A', R', D')$ is equal to C_i, by checking that $(A', R') = (A_1, R_1)$ and $e(A_1, R_1) = e(A_i, D')$. Thus, $\mathcal{B}$ can indeed test that the query Q^* is i-incriminating.
   In this case $\mathcal{B}$ knows that the value D_i inside the ciphertext C_i is equal to $D_i = R_1^{a_1/a_i}$. Since

$R_1 = R$, $a_1 = a$ and $\mathcal{B}$ knows a_i, $\mathcal{B}$ can compute $D_i^{a_i} = R^a$, and then test if

$$e(D_i^{a_i}, B) \stackrel{?}{=} g_1$$

In either case, $\mathcal{B}$ will abort the entire simulation and output guess $b' = 0$ if and only if the test above passes.

To explain $\mathcal{B}$'s behavior, notice that $e(D_i^{a_i}, B) = e(R^a, g^b) = e(g, g)^{abr}$. Hence, if $g_1 = e(g, g)^{abr}$ the test always passes, and if g_1 is random, it almost never passes.

4. **Stuck/Complete Simulation.** When $\mathcal{B}$ guesses the "incriminating index" i correctly, we know that $\mathcal{B}$ will encounter the incriminating query Q^*, and hence output the guess b', before it encounters any of the HProve(j, C_j, x) or OutDel$(j, C_j, *)$ queries that it cannot simulate. However, when $\mathcal{B}$'s guess for i is wrong, we could encounter such a query, or perhaps run $\mathcal{A}$ to completion (say, when G does not happen). In this case, $\mathcal{B}$ will output a random guess b'.

5. **"IN" Delegation Oracle InDel.** The oracle InDel(a', C', j) is identical for $j \neq i$ to what was done before by $\mathcal{B}$. Namely, if valid, it simply returns $O_a((D')^{a^*/\alpha_i})$, where $C' = (A', R', D', \sigma')$. For $j = i$, we know the secret key a_i, so we can simply evaluate InDel honestly using a' and a_i.

6. **Honest Delegation Oracle HDel.** When $\mathcal{A}$ calls HDel$(j_1, (A', R', D', \sigma'), j_2)$, $\mathcal{B}$ will first do the delegation and signature checks, rejecting if they fail. Otherwise, if $i \notin \{j_1, j_2\}$, $\mathcal{B}$ does the same thing as $\mathcal{B}$, returning $(D')^{a_i/a_j} = (D')^{\alpha_i/\alpha_j}$. Otherwise, either $j_1 = i$ or $j_2 = i$. We treat them differently:

   - If $\mathcal{A}$ calls HDel$(i, (A', R', D', \sigma'), j)$ where $j \neq i$, $\mathcal{B}$ first checks if $(A', R', D') = C_i$, which means $(A', R') = (A_1, R_1)$ and $e(A_1, R_1) = e(A_i, D')$. If this is the case, $\mathcal{B}$ knows this is an i-incriminating query, and will process it, as explained above.

Otherwise, $\mathcal{B}$ is supposed to return the value

$$(D')^{a_i/a_j} = ((D')^{a_i/\alpha_j})^{1/a} = \mathcal{O}_a((D')^{a_i/\alpha_j})$$

where $\mathcal{B}$ knows a_i and α_j. Thus, $\mathcal{B}$ can simply return $\mathcal{O}_a((D')^{a_i/\alpha_j})$ using its own oracle. To put it differently, we can pretend that this HDel query is actually an InDel query with an adversarial key a_i.

- If $\mathcal{A}$ calls HDel$(j, (A', R', D', \sigma'), i)$ where $j \neq i$, then we know we have not yet reached the i-incriminating query (assuming the guess for i is correct, else we don't care). This means that the ciphertex
  $(A', R', D', \sigma') \neq C_j$, or else the answer to this query will be equal to C_i, contradicting the fact that C_i has no incoming delegation edges. In this case we notice that even though key A_i is supposed to be honest, *we can pretend that a_i is an adversarial key, and simulate HDel$(j, (A', R', D', \sigma'), i)$ as if it is a call to OutDel$(j, (A', R', D', \sigma'), a_i)$.*
  Specifically, $\mathcal{B}$ can perform the delegation check
  $e(A', R') \stackrel{?}{=} e(A_j, D')$ and the signature check
  $e(H'(A', R'), R') \stackrel{?}{=} e(\sigma', g)$ himself (by evaluating H' according to the standard strategy). Moreover, we know that $(A', R') \neq (A_1, R_1)$ in this case, because $(A', R', D', \sigma') \neq C_j$ and the signature/delegation checks worked. Thus, in our simulation of H' for the value $H'(A', R')$, we set it to $H'(A', R') = (A')^{\gamma'}$ for some random γ' known to $\mathcal{B}$. $\mathcal{B}$ will combine this γ' with the values of signature σ' and secret key a_i known to $\mathcal{B}$, and return

  $$\text{HDel}(j, (A', R', D', \sigma'), i) := (\sigma')^{(a_i\gamma')^{-1} \bmod p}$$

  The proof of correctness is the same as before, and is omitted.

Analysis of Reduction. The analysis of the reduction is partitioned as to whether $\mathcal{B}$ managed

to reach the i-incriminating query Q^* before being stuck with the simulation. Let us call this event I, and notice that it happens at least when G happens and $\mathcal{A}$'s guess i correctly, so

$$\Pr[]I] \geqslant \frac{\Pr[]G]}{u-1} = \frac{\Pr[]G]}{\text{poly}(k)}$$

When I happens, $\mathcal{B}$'s advantage is at least $1 - 1/p = 1 - \text{negl}(\kappa)$. Otherwise, $\mathcal{B}$ simply outputs a random b', achieving advantage $1/2$. This gives overall advantage of $\mathcal{B}$ equal to

$$\Pr[]b' = b] \geqslant \Pr[]I] \cdot (1 - \text{negl}(\kappa)) + (1 - \Pr[]I]) \cdot \frac{1}{2} = \frac{1}{2} + \frac{\Pr[]G]}{\text{poly}(k)}$$

And since we assume that the iBDDH assumption is true, we know $\mathcal{B}$'s advantage must at most $1/2 + \text{negl}(\kappa)$, which means $\Pr[]G] = \text{negl}(\kappa)$, completing the proof of Lemma 8.22. □

8.5.4 Construction of One-time Delegatable EVRF

Note that the bidirectional-delegation security of Construction 8.6 relied on a very strong inversion-oracle BDDH (iBDDH) assumption, which is interactive and not well studied. For applications where we only guarantee security after a single delegation, we could prove bidirectional-delegation under a much reasonable extended BDDH (eBDDH) assumption. More precisely, any party P is "safe" to do any number of "out-delegations" to other, potentially untrusted parties P', but should only accept "in-delegation" from such an untrusted P' only if the delegated ciphertext C' was created directly for P' (and not delegated to P' from somewhere else).

More formally, the one-time delegation scheme we present here is identical to the unidirectional delegation scheme from the previous section, except we replace the "delegation check"

$$e(A, R) \stackrel{?}{=} e(A_1, D_1)$$

by a stricter "equality check":

$$(A, R) \stackrel{?}{=} (A_1, D_1)$$

which means that the ciphertext C_1 was directly created for public key $A_1 = A$. We call the resulting 1-time-delegatable construction DEVRF_3. We will now show that DEVRF_3 satisfies bidirectional-delegation security, but now under a much weaker (non-interactive) eBDDH assumption:

Theorem 8.24. *The* one-time *delegatable DEVRF_3 above satisfies the* ***Bi-$-Core*** *property under the* eBDDH *assumption in the random oracle model. It satisfies the strongest possible legality condition for the attacker (see Definition 8.21).*

Proof. The proof of 1-time delegation is largely similar to the proof of Theorem 8.20. As in that proof, we start with the simpler setting of standard legality condition on $\mathcal{A}$, from Definition 8.17, and then generalize it to the stronger legality in Definition 8.21.

Original Legality Condition. Recall, our goal is to construction a reduction $\mathcal{B}$ which uses the attacker $\mathcal{A}$, except must $\mathcal{B}$ must solve the harder eBDDH problem, as opposed to iBDDH. Namely, $\mathcal{B}$ is given (g, W, A, B, R, g_1), where $W = g^{1/a}$, as is no longer given full inversion oracle $\mathcal{O}_a(\cdot)$. Recall also that, in the proof of Theorem 8.20, the only place where $\mathcal{B}$ used the inversion oracle was to simulate $\text{InDel}(a', C', i)$ oracle.

Our main idea is to use the fact that in the 1-time delegation scenario, each valid ciphertext $C' = (A', R', D', \sigma')$ submitted to OutDel must have $A' \in \{A_1, \ldots, A_u\}$, while each such ciphertext submitted to InDel must have $A' \notin \{A_1, \ldots, A_u\}$ (or else the attacker breaks the discrete log of some unknown honest key A_i). Thus, if previously we only used the signature σ to help us simulate OutDel queries, by effectively extracting the value $\text{DH}(A', R')$ for the submitted ciphertext C', now we can use it instead for helping simulate InDel queries as well, since those queries operate on *disjoint sets of public keys A*.

More concretely, our new attacker $\mathcal{B}(g, W, A, B, R, g_1)$ for eBDDH will operate identically with

the previous attacker in the proof of Theorem 8.20 (which used the inversion-oracle), except with the following changes:

1. Initialization of $\mathcal{A}$, challenge ciphertext C_1, challenge output y, and oracles Reg, H, HDel, and HProve are identical.

2. **Hash Queries to H'.** Recall, our previous attacker set $H'(A_1, R_1) = g^{\gamma_1}$ for random $\gamma_1 \in_r Z_p$, but all other queries $H'(A', R') = (A')^{\gamma'}$ for fresh random $\gamma' \in_r Z_p$. Now, we still set $H'(A_1, R_1) = g^{\gamma_1}$, but set other queries $(A', R') \neq (A_1, R_1)$ as follows:

   - If $A' \in \{A_1, \ldots, A_u\}$, set $H'(A', R') = (A')^{\gamma'}$ for fresh random $\gamma' \in_r Z_p$, as before.
   - Otherwise, set $H'(A', R') = W^{\gamma}$ for fresh random $\gamma \in_r Z_p$, where $W = g^{1/a}$ comes from the input of $\mathcal{B}$.

3. **"OUT" Delegation Oracle OutDel.** When $\mathcal{A}$ calls OutDel$(i, (A', R', D', \sigma'), a^*)$, for any secret key a^*, any such (allowed) query must have $A' = A_i$, by our stricter delegation check. Hence, the oracle $H'(A', R') = (A')^{\gamma'}$, as in the proof of Theorem 8.20 (this uses the fact $(A', R') \neq (A_1, R_1)$, as this query is not allowed by legality of $\mathcal{A}$, even if $i = 1$.) Hence, we can use the same strategy as before. Namely, the value $\mathcal{B}$ can return the value

$$\text{OutDel}(i, (A', R', D', \sigma'), a^*) := (\sigma')^{(a^*\gamma')^{-1} \bmod p}$$

   as before, as this value is equal to $(D')^{a_i/a^*}$. The proof of this is the same as in Theorem 8.19.

   c

4. **"IN" Delegation Oracle InDel.** When $\mathcal{A}$ calls InDel$(a', (A', R', D', \sigma'), i)$, for any secret key a', we first check if $a' \in \{a_1, \ldots, a_u\}$, by checking that $g^{a'} \stackrel{?}{=} A_i$ for all i. If this is the case, $\mathcal{B}$ can compute $a = a'/\alpha_i$, which trivially allows

it to win its game by checking whether $g_1 \stackrel{?}{=} e(B, R)^a$.

Otherwise, we know that $H'(A', R') = W^\gamma$ for some random γ. From delegation check, we also know that $A' = g^{a'}$ and $R' = D'$, and from the signature check we know that $e(H(A', R'), R') = e(\sigma', g)$. This means

$$e(\sigma', g) = e(H(A', R'), R') = e(W^\gamma, R') = e((R')^{\gamma/a}, g)$$

This implies that $(R')^{1/a} = (\sigma')^{1/\gamma}$. But $\mathcal{A}$ expects to see

$$(D')^{a'/a_i} = (R')^{a'/(\alpha_i a)} = ((R')^{1/a})^{a'/\alpha_i} = ((\sigma')^{1/\gamma})^{a'/\alpha_i} = (\sigma')^{a'/(\gamma\alpha_i)}$$

Thus, $\mathcal{B}$ can complete the simulation by responding with

$$\textsc{InDel}(a', (A', R', D', \sigma'), i) = (\sigma')^{a'/(\gamma\alpha_i)}$$

This completes our reduction for the basic legality condition.

Stronger Legality Condition. Finally, we show how to extend our proof to the optimal legality condition given in Definition 8.21. This will be done very similar to the proof of Theorem 8.20, but slightly simpler due to the more limited structure of the delegation graph M used in the proof of Theorem 8.20.

In particular, recall "special" ciphertexts $C_i = (A_1, R_1, D_i, \sigma)$, where $D_i = R_1^{a_i/a_1}$ and $\sigma = H'(A_1, R_1)^{r_1}$. Except for $i = 1$, none of these ciphertexts can be delegated. Thus, the "gap" event G between the 2 legality conditions of $\mathcal{A}$ becomes simply:

- $\mathcal{A}$ made a call to $\textsc{HProve}(i, C_i, x)$ for some $i > 1$, where x is the challenge input produced by $\mathcal{A}_1$, but did not query $\textsc{HDel}(1, C_1, i)$.

If G does not happen, this corresponds to the original legality condition, for which we already

finished the proof. Thus, it remains to show that for any PPT attacker $\mathcal{A}$, we have $\Pr[[]G] = \text{negl}(\kappa)$. Once again, this is proven similarly to the corresponding proof of Lemma 8.22 in the proof of Theorem 8.20.

In particular, we will build a reduction $\mathcal{B}$ to eBDDH from any attacker $\mathcal{A}$ which triggered the gap event G. Since our 1-time delegatable scheme, DEVRF_3 is really a special case of the general scheme DEVRF_2 considered in proof of Theorem 8.20, our reduction $\mathcal{B}$ can be identical to the "old" reduction — call it $\mathcal{B}'$ — used in the proof of Lemma 8.22, except we need to make sure we no longer need to use the inversion oracle $\mathcal{O}_a(\cdot)$, and only use the value $W = g^{1/a}$. Fortunately, we can do it using the same technique as in the proof of the original legality condition, by basically choosing a random index $i > 1$ (hoping it corresponds to the incriminating query of $\mathcal{A}$), and effectively treating the known key a_i as adversarial. We highlight the differences here:

1. **Registration Oracle Reg.** We simulate exactly like the previous reduction $\mathcal{B}'$. We choose a random index $i > 1$ at random, and set all public keys $A_j = A_1^{\alpha_j}$ for $j \neq i$ for random α_j. However, for the i-th secret key $\mathcal{B}$ will chose a random key $a_i \in_r \mathbb{Z}_p$ and honestly set $A_i = g^{a_i}$. In other words, $\mathcal{B}$ will actually know the i-th secret key.

2. **New Simulation of H'.** As done earlier in the section, we set $H'(A', R')$ as follows:

   - If $(A', R') = (A_1, R_1)$, set $H'(A_1, R_1) = g^{\gamma_1}$, for random $\gamma_1 \in_r Z_p$
   - If $A' \in \{A_j \mid j \neq i\}$, set $H'(A', R') = (A')^{\gamma'}$ for fresh random $\gamma' \in_r Z_p$.
   - Otherwise, set $H'(A', R') = W^{\gamma}$ for fresh random $\gamma \in_r Z_p$.
     (Note, this critically includes the values $H'(A_i, R')$.)

   Due to the 1-time delegation nature of DEVRF_3, this restriction on H' still allows us to correctly simulate all $\text{OutDel}(j, C', *)$ calls for $j \neq i$, as they will use $H'(A_j, R') = (A_j)^{\gamma'}$. While for $j = i$ we can simply use the secret key a_i.

3. **Places Previous $\mathcal{B}'$ used $O_a(\cdot)$.** Examining the proof of Lemma 8.22, there were only two places where $\mathcal{B}'$ used the inversion oracle $O_a(\cdot)$:

   - **Case 1:** When $\mathcal{A}$ called $\textsc{InDel}(a', (A', R', D', \sigma'), j)$. For $j = i$, our new reduction $\mathcal{B}$ can simply use a_i, as did $\mathcal{B}$. However, for $j \neq i$, instead of calling (no longer present) oracle $O_a((D')^{a^*/\alpha_j})$, we use the same strategy we used at the beginning of this section to extract the answer from the signature σ' supplied by the attacker. Namely, we know that $g^{a'} = A'$, $R' = D'$, $H(A', R') = W^\gamma$ in our simulation (as otherwise $a' = a_j = \alpha_j a$, for $j \neq i$, which allows us to break eBDDH trivially), and $e(H(A', R'), R') = e(\sigma', g)$. This means

   $$e(\sigma', g) = e(H(A', R'), R') = e(W^\gamma, R') = e((R')^{\gamma/a}, g)$$

   This implies that $(R')^{1/a} = (\sigma')^{1/\gamma}$. But $\mathcal{A}$ expects to see

   $$\begin{aligned}(D')^{a'/a_j} &= (R')^{a'/(\alpha_j a)} \\ &= ((R')^{1/a})^{a'/\alpha_j} \\ &= ((\sigma')^{1/\gamma})^{a'/\alpha_j} = (\sigma')^{a'/(\gamma\alpha_j)}\end{aligned}$$

   Thus, $\mathcal{B}$ can complete the simulation by responding with

   $$\textsc{InDel}(a', (A', R', D', \sigma'), j) = (\sigma')^{a'/(\gamma\alpha_j)}$$

   - **Case 2:** When $\mathcal{A}$ called $\textsc{HDel}(i, (A', R', D', \sigma'), j)$ for $j \neq i$. However, by 1-time delegation check we know that $A' = A_i$, which means $H(A_i, R') = W^\gamma$ as well. And hence we can use exactly the same strategy as in the previous **Case 1**, effectively treating the known a_i as the adversarial key and returning

   $$\textsc{HDel}(i, (A', R', D', \sigma'), j) = (\sigma')^{a_i/(\gamma\alpha_j)}$$

This completes the new reduction $\mathcal{B}$ to the eBDDH problem, and the overall proof of the stronger legality condition of DEVRF_3.

□

We stress that our 1-time delegatable scheme could in principle be delegated further, if the stricter delegation check $(A, R) \stackrel{?}{=} (A_1, D_1)$ is replaced by the original check $e(A, R) \stackrel{?}{=} e(A_1, D_1)$. However, by doing so the party receiving the EVRF from some untrusted source must rely on the stronger iBDDH complexity assumption.

Part IV

Updatable Public Key Encryption

9 | Updatable Public Key Encryption in the Standard Model

This chapter is based on joint work with Yevgeniy Dodis and Daniel Wichs that appeared in TCC 2021 [DKW21]. Passages are taken verbatim from the full version of this paper [DKW22b]. This chapter considers the primitive known as updatable public key encryption (UPKE).

9.1 Preliminaries

Theorem 9.1 (Leftover Hash Lemma)**.** *Fix $\varepsilon > 0$. Let X be a random variable on $\{0,1\}^n$ with conditional min-entropy $\mathrm{H}_\infty(X|E) \geq k$. Let $\mathcal{H} = \{\mathcal{H}_n\}_{n\in\mathbb{N}}$ where $\mathcal{H}_n = \{h_s\}_{s\in\{0,1\}^d}$ for all n, be a universal hash family with output length $m \leq k - 2\log(1/\varepsilon)$. Then,*

$$(h_{U_d}(X), U_d, E) \approx_\varepsilon (U_m, U_d, E)$$

Lemma 9.2 (Smudging Lemma [AJL+12])**.** *Let $B_1 = B_1(\kappa)$ and $B_2 = B_2(\kappa)$ be positive integers and let $e_1 \in [-B_1, B_1]$ be a fixed integer. Let $e_2 \leftarrow\!\!\$\ [-B_2, B_2]$ be chosen uniformly at random. Then the distribution of e_2 is statistically indistinguishable from $e_1 + e_2$ as long as $B_1/B_2 = \mathrm{negl}(\kappa)$.*

Definition 9.3 (The Decisional Diffie Hellman Assumption (DDH))**.** Let $\mathcal{G}$ be a probabilistic polynomial-time "group generator" that, given as a parameter 1^κ where κ is the security param-

eter, outputs the description of a group $\mathbb{G}$ that has prime order $p = p(\kappa)$. The decisional Diffie Hellman (DDH) assumption for $\mathcal{G}$ says that the following two ensembles are computationally indistinguishable:

$$\{(g_1, g_2, g_1^r, g_2^r) : g_i \leftarrow \mathbb{G}, r \leftarrow \mathbb{Z}_p\} \approx_c \{(g_1, g_2, g_1^{r_1}, g_2^{r_2}) : g_i \leftarrow \mathbb{G}, r_i \leftarrow \mathbb{Z}_p\}$$

A lemma of Naor and Reingold [NR97] generalizes the above assumption for $m > 2$ generators.

Lemma 9.4 ([NR97])**.** *Under the DDH assumption on $\mathcal{G}$,*

$$\{(g_1, \ldots, g_m, g_1^r, \ldots, g_m^r) : g_i \leftarrow \mathbb{G}, r \leftarrow \mathbb{Z}_p\} \approx_c \{(g_1, \ldots, g_m, g_1^{r_1}, \ldots, g_m^{r_m}) : g_i \leftarrow \mathbb{G}, r_i \leftarrow \mathbb{Z}_p\}$$

Definition 9.5 (Learning with Errors Assumption (LWE))**.** Consider integers $n,\ m,\ q$ and a probability distribution χ on $\mathbb{Z}_q$, typically taken to be a normal distribution that has been discretized. Then, the LWE assumption states that the following two ensembles are computationally indistinguishable:

$$\{\boldsymbol{A}, \boldsymbol{A}^T\boldsymbol{x} + \boldsymbol{e} : \boldsymbol{A} \leftarrow_\$ \mathbb{Z}_q^{n\times m}, \boldsymbol{x} \leftarrow_\$ \mathbb{Z}_q^n, \boldsymbol{e} \leftarrow_\$ \chi^m\} \approx_c \{\boldsymbol{A}, \boldsymbol{v} : \boldsymbol{A} \leftarrow_\$ \mathbb{Z}_q^{n\times m}, \boldsymbol{v} \leftarrow_\$ \mathbb{Z}_q^m\}$$

9.2 Updatable Public Key Encryption (UPKE)

Jost *et al.* [JMM19] introduced the notion of an Updatable Public Key Encryption (UPKE). This definition was later modified by the work of Alwen *et al.* [ACDT20]. Below, we present our variant of the UPKE.

Definition 9.6. An updatable public key encryption (UPKE) scheme is a set of five polynomial-time algorithms UPKE = (U-PKEG, U-Enc, U-Dec, Upd-Pk, Upd-Sk) with the following syntax:

- **Key generation**: U-PKEG takes as parameter 1^κ where κ is the security parameter and

outputs a fresh secret key sk_0 and a fresh initial public key pk_0.

- **Encryption**: U-Enc receives a public key pk and a message m to produce a ciphertext c.
- **Decryption**: U-Dec receives a secret key sk and a ciphertext c to produce message m.
- **Update Public Key**: Upd-Pk receives a public key pk to produce an update ciphertext up and a new public key pk′.
- **Update Secret Key**: Upd-Sk receives an update ciphertext up and secret key sk to produce a new secret key sk′.

We require the following security properties from this primitive:

Correctness. Let (sk_0, pk_0) be the output of U-PKEG. For any sequence of randomness $\{r_i\}_{i=1}^{q}$, define the sequence of public keys and secret keys $\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i=1}^{q}$ as follows:

for $i = 1$ **to** q

$(\mathsf{up}_i, \mathsf{pk}_i) \leftarrow \mathsf{Upd\text{-}Pk}(\mathsf{pk}_{i-1}; r_i)$

$\mathsf{sk}_i \leftarrow \mathsf{Upd\text{-}Sk}(\mathsf{sk}_{i-1}, \mathsf{up}_i)$

Then, UPKE is correct if for any message m and for any $j \in [q]$,

$$\Pr[\mathsf{U\text{-}Dec}(\mathsf{sk}_j, \mathsf{U\text{-}Enc}(\mathsf{pk}_j, m)) = m] = 1 \,.$$

9.2.1 IND-CR-CPA Security of UPKE

In this section, we define the security game. We will called this the IND-CR-CPA Security which is meant to capture INDistinguishibility under Chosen Randomness Chosen Plaintext Attack. Largely similar to the CPA security game, this also additionally allows the adversary to choose the randomness used to update the keys which is modeled by the following oracle access:

- $\mathcal{O}_{upd}(\cdot)$: $\mathcal{A}$ provides its choice of randomness r_i. The Challenger performs the following actions:

  $(\mathsf{up}_{i+1}, \mathsf{pk}_{i+1}) \leftarrow \mathsf{Upd\text{-}Pk}(\mathsf{pk}_i; r_i)$
  $\mathsf{sk}_{i+1} \leftarrow \mathsf{Upd\text{-}Sk}(\mathsf{sk}_i, \mathsf{up}_{i+1})$.
  $i = i + 1$

In this game, i is the update counter, initialized to 0. For any adversary $\mathcal{A}$ with running time t we consider the IND-CR-CPA security game:

- Sample $(\mathsf{sk}_0, \mathsf{pk}_0) \leftarrow \mathsf{U\text{-}PKEG}(1^\kappa)$, $b \leftarrow\!\!\$\ \{0, 1\}$. Set $i = 0$.
- $(m_0^*, m_1^*, state) \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}_{upd}(\cdot)}(\mathsf{pk}_0)$
- Compute $c^* \leftarrow\!\!\$\ \mathsf{U\text{-}Enc}(\mathsf{pk}_{q'}, m_b^*)$ where $i = q'$ is the current time period.
- $state \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}_{upd}(\cdot)}(c^*, state)$
- Choose uniformly random r^* and then compute

$$(\mathsf{up}^*, \mathsf{pk}^*) \leftarrow \mathsf{Upd\text{-}Pk}(\mathsf{pk}_q; r^*);\ \mathsf{sk}^* \leftarrow \mathsf{Upd\text{-}Sk}(\mathsf{sk}_q, \mathsf{up}^*) .$$

  where $i = q$ is the current time period.

- $b' \leftarrow\!\!\$\ \mathcal{A}(\mathsf{pk}^*, \mathsf{sk}^*, \mathsf{up}^*, state)$.
- $\mathcal{A}$ wins the game if $b = b'$. The advantage of $\mathcal{A}$ in winning the above game is denoted by $\mathrm{Adv}^{\mathsf{UPKE}}_{\mathrm{crcpa}}(\mathcal{A}) = |\Pr[b = b'] - \frac{1}{2}|$.

Definition 9.7. An updatable public-key encryption scheme UPKE is IND-CR-CPA -secure if for all PPT attackers $\mathcal{A}$, its advantage $\mathrm{Adv}^{\mathsf{UPKE}}_{\mathrm{crcpa}}(\mathcal{A})$ is negligible.

Remark 18 (Comparison of the Security Models.)**.** The work of Jost *et al.* [JMM19] defined a notion which had an update procedure not specific to any public key. This was designed to support multiple instances, i.e. multiple key pairs, and where the offset generated by the public update could be applied to many public keys. While we consider the simpler setting of only one instance, which is also reflected in our syntax, we believe that our constructions trivially satisfy the stronger security model proposed by [JMM19]. Our model also allows for $q \neq q'$, i.e., for the adversary to issue a challenge in one time period and corrupt in another time period. However, without loss of generality, we give the attacker the final secret key sk^* immediately following the honest post-challenge key update (at period q'), as this gives the most amount of information to the attacker.

Our definition is a generalization of the model proposed by Alwen *et al.* [ACDT20]: their notion forced an update of the keys after every encryption query, while ours separates the two processes for more flexibility.

9.3 Key-Dependent-Message-Secure Encryption Scheme

Let us recall the definition of a public-key encryption scheme.

Definition 9.8. An encryption scheme is a set of three polynomial-time algorithms PKE = (Gen, Enc, Dec) with the following syntax:

- **Key generation:** Gen receives 1^κ where κ is the security parameter and outputs a fresh secret sk and outputs a fresh public key pk.
- **Encryption:** Enc receives a public key pk and a message m to produce a ciphertext c.
- **Decryption:** Dec receives a secret key sk and a ciphertext c to produce message m.

Correctness. The correctness of an encryption scheme is such that $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa), \forall m \in \mathcal{M}$,

$$\Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m] = 1$$

CS+LR Security. For any PPT adversary $\mathcal{A}$ we consider the following security game:

- Sample $(\mathsf{sk}, \mathsf{pk}) \leftarrow\!\!\$\ \mathsf{Gen}(1^\kappa), b \leftarrow\!\!\$\ \{0,1\}$.
- $L, f, m_0, m_1 \leftarrow\!\!\$\ \mathcal{A}(\mathsf{pk})$ where L is the leakage function chosen by $\mathcal{A}$, m_0, m_1 are the challenge messages, and f is the function of the secret key that $\mathcal{A}$ wants to receive as encryption. L defines the leakage resilience and f defines the KDM security.
- Compute $C \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{pk}, m_b), C' \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{pk}, f(\mathsf{sk}))$[1].
- $b' \leftarrow\!\!\$\ \mathcal{A}(c_0, c_1, L(\mathsf{sk}))$.
- $\mathcal{A}$ wins the game if $b = b'$. The advantage of $\mathcal{A}$ in winning the above game is denoted by $\mathsf{Adv}^{\mathsf{PKE}}_{\mathsf{KDM}}(\mathcal{A}) = |\Pr[b = b'] - \frac{1}{2}|$.

Definition 9.9. A public-key encryption scheme PKE is λ-CS+LR-secure if for all PPT attackers $\mathcal{A}$, and leakage functions L such that $\mathrm{H}_\infty(\mathsf{sk}|L(\mathsf{sk})) \geq |\mathsf{sk}| - \lambda$, its advantage $\mathsf{Adv}^{\mathsf{PKE}}_{\mathsf{cs+lr}}(\mathcal{A})$ is negligible.

9.4 DDH Based Construction

This section presents construction from the DDH Assumption. We begin by presenting a slightly modified version of the PKE Scheme proposed by Boneh *et al.* [BHHO08] in section 9.4.1. This scheme was shown to be independently circular secure and leakage resilient. We also show that the scheme is CS+LR secure in section 9.4.2. We then present our construction of a UPKE

[1] In our security proofs, the function f will be applied to each bit of the secret key.

Protocol BHHO Cryptosystem

$\underline{\mathsf{Gen}(1^\kappa)}$

Sample $\boldsymbol{s} = (s_1, \ldots, s_\ell) \leftarrow_s \{0,1\}$ and $g_1, \ldots, g_\ell \leftarrow_s \mathbb{G}$.
Compute $h = \prod_{i=1}^{\ell} g_i^{s_i}$.
return $\mathsf{sk} = \boldsymbol{s} \in \mathbb{Z}_p^\ell$, $\mathsf{pk} = (g_1, \ldots, g_\ell, h) \in \mathbb{G}^{\ell+1}$.

$\underline{\mathsf{Enc}(\mathsf{pk}, m \in \mathbb{G})}$

Parse $\mathsf{pk} = (g_1, \ldots, g_\ell, h)$
Sample $r \leftarrow_s \mathbb{Z}_p$
for $i = 1, \ldots, \ell$ **do**
  Compute $f_i = g_i^r$
return $C = (f_1, \ldots, f_\ell, c = h^r \cdot m) \in \mathbb{G}^{\ell+1}$

$\underline{\mathsf{Dec}(\mathsf{sk}, C)}$

Parse $C = (f_1, \ldots, f_\ell, c = h^r \cdot m)$ and $\mathsf{sk} = \boldsymbol{s} = (s_1, \ldots, s_\ell) \in \mathbb{Z}_p^\ell$
Compute $m' = c \cdot \left(\prod_{i=1}^{\ell} f_i^{s_i}\right)^{-1}$
return m'

Construction 9.1: A modified version of the BHHO Cryptosystem where the bits of the secret key are not encoded as group elements. Let κ be the the security parameter. Let $\mathcal{G}$ be a probabilistic polynomial-time "group generator" that takes as input 1^κ and outputs the description of a group $\mathbb{G}$ with prime order $p = p(\kappa)$ and g is a fixed generator of $\mathbb{G}$.

scheme (section 9.4.3), extended from the PKE scheme. We finally prove that the UPKE scheme is IND-CPA secure in section 9.4.4.

9.4.1 The BHHO Cryptosystem

In this section, we present a modified version of the original BHHO Cryptosystem. This is presented as Construction 9.1.

Correctness. Let $m \in \mathbb{G}$. $\mathsf{Enc}(\mathsf{pk}, m) = (f_1 = g_1^r, \ldots, f_\ell = g_\ell^r, c = h^r \cdot m)$. $\mathsf{Dec}(\mathsf{sk}, f_1, \ldots, f_\ell, c)$ outputs:

$$c \cdot \left(\prod_{i=1}^{\ell} f_i^{s_i}\right)^{-1} = h^r \cdot m \left(\prod_{i=1}^{\ell} f_i^{s_i}\right)^{-1} = \left(\prod_{i=1}^{\ell} g_i^{s_i}\right)^r \cdot m \cdot \left(\prod_{i=1}^{\ell} (g_i^r)^{s_i}\right)^{-1} = m.$$

9.4.2 CS+LR Security of BHHO Cryptosystem

In this section, we provide proof of the combined circular security and leakage resilience of the BHHO Cryptosystem. Formally, we will prove the following theorem:

Theorem 9.10. *Under the DDH Assumption, Construction 9.1 is λ-CS+LR secure for leakage $\lambda = \ell - 2\log p - \omega(\log\kappa)$.*

However, before we can prove the theorem, we will prove that there exists an algorithm $\mathsf{Enc}'(\mathsf{pk}, i)$ such that

$$(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, g^{s_i}), \boldsymbol{s}) \approx_c (\mathsf{pk}, \mathsf{Enc}'(\mathsf{pk}, i), \boldsymbol{s}).$$

Consider the following definition of Enc':

$$\mathsf{Enc}'(\mathsf{pk}, i) = (f_1 = g_1^r, \ldots, f_{i-1} = g_{i-1}^r, f_i = g_i^r/g, f_{i+1} = g_{i+1}^r, \ldots, f_\ell = g_\ell^r, h^r)$$

We will first show that this ciphertext decrypts correctly to g^{s_i}.

$$\begin{aligned}\mathsf{Dec}(\boldsymbol{s}, f_1, \ldots, f_\ell, c = h^r) &= h^r \cdot \left(\prod_{i=1}^{\ell} f_i^{s_i}\right)^{-1} \\ &= h^r \left(\prod_{i=1}^{\ell} g_i{}^{s_i}\right)^{-r} g^{s_i} = h^r \cdot h^{-r} \cdot g^{s_i} = g^{s_i}\end{aligned}$$

Lemma 9.11. *Under the DDH Assumption,* $(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, g^{s_i}), \boldsymbol{s}) \approx_c (\mathsf{pk}, \mathsf{Enc}'(\mathsf{pk}, i), \boldsymbol{s})$ *where* $(\mathsf{pk}, \boldsymbol{s}) \leftarrow\!\!\$\ \mathsf{Gen}(1^\kappa)$

Proof. We will prove the lemma through a sequence of hybrids. This is a tabulated summary of the changes through the proof:

Hybrid	Hybrid Definition	Security
D_0	Enc is used to encrypt g^{s_i}	Identical
D_1	D_0 except $h^r \cdot g^{s_i}$ replaced with $\prod_{j=1}^{\ell} f_j^{s_j} \cdot g^{s_i}$	DDH
D_2	D_1 except each $f_j \leftarrow\!\!\$\ \mathbb{G}$	Identical
D_3	D_2 except f_i is replaced by f_i/g where $f_i \leftarrow\!\!\$\ \mathbb{G}$	DDH
D_4	D_3 except $f_j = g_j^r$ where $r \leftarrow\!\!\$\ \mathbb{Z}_p$	Identical
D_5	Enc′ is used to encrypt g^{s_i}	

Hybrid D_0. This is when Enc is used to encrypt g^{s_i}. It corresponds to the distribution:

$$(\mathsf{pk}, g_1^r, \ldots, g_\ell^r, h^r \cdot g^{s_i}, \boldsymbol{s} : r \leftarrow\!\!\$\ \mathbb{Z}_p)$$

Hybrid D_1. This is same as Hybrid D_0 where we replace h^r by the steps of the decryption algorithm. It corresponds to the distribution

$$\left(\mathsf{pk}, f_1 = g_1^r, \ldots, f_\ell = g_\ell^r, \prod_{j=1}^{\ell} f_j^{s_j} \cdot g^{s_i}, \boldsymbol{s} : r \leftarrow\!\!\$\ \mathbb{Z}_p\right)$$

The distributions D_0 and D_1 are identical for the same value of $r \leftarrow\!\!\$\ \mathbb{Z}_p$. Therefore, there is no distinguishing advantage for any adversary $\mathcal{A}$.

Hybrid D_2. In this case, we sample each $f_i \leftarrow_\$ \mathbb{G}$. This corresponds to the distribution:

$$\left(\mathsf{pk}, f_1, \ldots, f_\ell, \prod_{j=1}^{\ell} f_j^{s_j} \cdot g^{s_i}, \boldsymbol{s} : f_1, \ldots, f_\ell \leftarrow_\$ \mathbb{G}\right)$$

Claim 9.12. If DDH (as defined in Lemma 9.4) is hard for $\mathbb{G}$, then for every PPT $\mathcal{A}$, the advantage in distinguishing Hybrids D_1 and D_2 is negligible.

Proof. We will use an adversary $\mathcal{A}$ capable of distinguishing between the two distributions to create an adversary $\mathcal{B}$ that can win against the DDH Game. After receiving input from the challenger $(g_1, \ldots, g_\ell, f_1, \ldots, f_\ell)$, $\mathcal{B}$ generates $(\mathsf{pk}, \mathsf{sk} = \boldsymbol{s})$ and returns to $\mathcal{A}$: $(f_1, \ldots, f_\ell, \prod_{j=1}^{\ell} f_j^{s_j} \cdot g^{s_i}, \boldsymbol{s})$. It is easy to see that $\mathcal{B}$ perfectly simulates one of the hybrids based on the input it receives. This concludes the proof that $\mathcal{A}$ has negligible advantage in distinguishing the two hybrids. □

Hybrid D_3. The same distribution as Hybrid 2, except that f_i is replaced by f_i/g.

$$\left(\mathsf{pk}, f_1, \ldots, f_{i-1}, f_i/g, f_{i+1}, \ldots, f_\ell, \prod_{j=1}^{\ell} f_j^{s_j} \cdot g^{s_i}, \boldsymbol{s} : f_1, \ldots, f_\ell \leftarrow_\$ \mathbb{G}\right)$$

We know that for fixed g, f_i/g is indistinguishable from f_i where $f_i \leftarrow_\$ \mathbb{G}$. Therefore, the distributions are identical and $\mathcal{A}$ has no advantage in distinguishing the two distributions.

Hybrid D_4. This is corresponding to the distribution where $f_j = g_j^r$ where $r \leftarrow_\$ \mathbb{Z}_p$.

$$\left(\mathsf{pk}, g_1^r, \ldots, g_{i-1}^r, g_i^r/g, g_{i+1}^r, \ldots, g_\ell^r, \prod_{j=1}^{\ell} f_j^{s_j} \cdot g^{s_i}, \boldsymbol{s} : r \leftarrow_\$ \mathbb{Z}_p\right)$$

Claim 9.13. If DDH (as defined in Lemma 9.4) is hard for $\mathbb{G}$, then for every PPT $\mathcal{A}$, the advantage in distinguishing Hybrids D_3 and D_4 is negligible.

The proof of this claim is similar to the proof of the earlier claim.

Hybrid 5. This is corresponding to the distribution $(\mathsf{Enc}'(\mathsf{pk}, i), \boldsymbol{s})$, $f_i = g_i^r / g$ where $r \leftarrow\!\!\$\ \mathbb{Z}_p$.

$$(\mathsf{pk}, f_1 = g_1^r, \ldots, f_{i-1} = g_{i-1}^r, f_i = g_i^r / g, f_{i+1} = g_{i+1}^r, \ldots, f_\ell = g_\ell^r, h^r, \boldsymbol{s} : r \leftarrow\!\!\$\ \mathbb{Z}_p)$$

It is clear that the input distribution in Hybrids D_4 and D_5 are identical for the same r and $\mathcal{A}$ has no advantage in distinguishing the two distributions. This is because: $\prod_{j=1}^{\ell} f_j^{s_j} \cdot g^{s_i} = \prod_{j \neq i} g_j^{r s_j} \cdot g_i^{r s_i} / g^{s_i} \cdot g^{s_i} = (\prod_{j=1}^{\ell} g_j^{s_j})^r = h^r$.

Therefore, we have shown that $(\mathsf{Enc}(\mathsf{pk}, g^{s_i}), \boldsymbol{s}) \approx_c (\mathsf{Enc}'(\mathsf{pk}, i), \boldsymbol{s})$. □

Further, note that each s_i is independently chosen. Additionally, each encryption/fake-encryp-tion chooses its own independent randomness r. Therefore, we can independently replace each $\mathsf{Enc}(\mathsf{pk}, g^{s_i})$ with $\mathsf{Enc}'(\mathsf{pk}, i)$, and the resulting encryption of secret key is computationally indistinguishable from the one computed by Enc'. This proof can be shown by a sequence of hybrids, replacing one encryption at a time. Therefore, as a corollary we get that:

Corollary 9.14. *Under the DDH Assumption,*

$$(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, g^{s_1}), \ldots, \mathsf{Enc}(\mathsf{pk}, g^{s_\ell}), \boldsymbol{s}) \approx_c (\mathsf{pk}, \mathsf{Enc}'(\mathsf{pk}, 1), \ldots, \mathsf{Enc}(\mathsf{pk}, \ell), \boldsymbol{s})$$

With this corollary, we can prove the original theorem:

Theorem 9.10. *Under the DDH Assumption, Construction 9.1 is λ-CS+LR secure for leakage $\lambda = \ell - 2\log p - \omega(\log \kappa)$.*

Proof. We will prove the same through a sequence of hybrids. This is a tabulated summary of the hybrids changes through the proof:

Hybrid	Hybrid Definition	Security
D_0	The Original CS+LR Security Game, Enc is used	Corollary 9.14
D_1	D_0 except Enc' is used	Identical
D_2	D_1 except except $h^r \cdot m_b$ replaced with $\prod_{j=1}^{\ell} f_j^{s_j} \cdot m_b$	DDH
D_3	D_2 except each f_i is replaced by $f_i \leftarrow\!\!\$\ \mathbb{G}$	Leftover Hash Lemma
D_4	D_3 except $\prod_{j=1}^{\ell} f_j^{s_j} \cdot m_b$ replaced by $U \leftarrow\!\!\$\ \mathbb{G}$	

Note that each of our hybrid distribution contains pk and $L(\mathsf{sk} = \boldsymbol{s})$ in its definition. We drop these terms from the definition for simplicity and merely focus on the two ciphertexts which undergo the bulk of the changes.

Hybrid D_0. The original CS+LR Game. In this hybrid, $\mathcal{A}$ receives the following distribution:

$$\left(C = (f_1 = g_1^r, \ldots, f_\ell = g_\ell^r, h^r \cdot m_b), C' = (\mathsf{Enc}(\mathsf{pk}, g^{s_1}), \ldots, \mathsf{Enc}(\mathsf{pk}, g^{s_\ell})) : r \leftarrow\!\!\$\ \mathbb{Z}_p\right)$$

Hybrid D_1. The CS+LR Game but with C' consisting of the "fake encryption" algorithm. This corresponds to the distribution:

$$\left(C = (f_1 = g_1^r, \ldots, f_\ell = g_\ell^r, h^r \cdot m_b), C' = (\mathsf{Enc}'(\mathsf{pk}, 1), \ldots, \mathsf{Enc}'(\mathsf{pk}, \ell)) : r \leftarrow\!\!\$\ \mathbb{Z}_p\right)$$

In Corollary 9.14 we showed that the two distribution were indistinguishable even when conditioned on the secret key $\boldsymbol{s}$. However, in the definition of D_0, D_1, we only provide partial leakage $L(\boldsymbol{s})$, and hence $\mathcal{A}$ has negligible advantage in distinguishing the two distributions.

Hybrid D_2. It is similar to hybrid D_1, but with $h^r \cdot m_b$ replaced by $\prod_{j=1}^{\ell} f_j^{s_j} \cdot m_b$. This is the following distribution:

$$\left(C = \left(f_1 = g_1^r, \ldots, f_\ell = g_\ell^r, \prod_{j=1}^{\ell} f_j^{s_j} \cdot m_b\right), C' : r \leftarrow\!\!\$\ \mathbb{Z}_p\right)$$

For the same r, the distributions from Hybrids D_2 and D_3 are identical. Therefore, $\mathcal{A}$ has no advantage in distinguishing the two hybrids.

Hybrid D_3. Similar to hybrid D_2, except each $f_i \leftarrow\!\!\$\ \mathbb{G}$. This is the following distribution:

$$\left(C = \left(f_1, \ldots, f_\ell, \prod_{j=1}^{\ell} f_j^{s_j} \cdot m_b\right), C' : f_1, \ldots, f_\ell \leftarrow\!\!\$\ \mathbb{G}\right)$$

Claim 9.15. If DDH is hard for $\mathbb{G}$, then for every PPT $\mathcal{A}$, the advantage in distinguishing hybrids D_2 and D_3 is negligible.

Proof. We will use an adversary $\mathcal{A}$ capable of distinguishing hybrids D_2 and D_3 to create $\mathcal{B}$ that can win against the DDH Game. $\mathcal{B}$ receives from the DDH Challenger: $(g_1, \ldots, g_\ell, f_1, \ldots, f_\ell)$. It chooses $\boldsymbol{s} \leftarrow\!\!\$\ \{0,1\}^\ell$ and sets $\mathsf{pk} = (g_1, \ldots, g_\ell, h)$ where $h = \prod_{i=1}^{\ell} g_i^{s_i}$ and sets $\mathsf{sk} = \boldsymbol{s}$. It then sends to $\mathcal{A}$: $(\mathsf{pk}, L(\mathsf{sk} = \boldsymbol{s}), C = (f_1, \ldots, f_\ell, \prod_{j=1}^{\ell} f_j^{s_j} \cdot m_b), C' = (\mathsf{Enc}'(\mathsf{pk}, 1), \ldots, \mathsf{Enc}'(\mathsf{pk}, \ell)))$. It is easy to see that $\mathcal{B}$ perfectly simulates the distributions of hybrids D_2 and D_3 based on the input it receives. It merely forwards $\mathcal{A}$'s guess as its own. This concludes the proof that $\mathcal{A}$ has a negligible advantage in distinguishing hybrids D_2 and D_3. □

Hybrid D_4. Replace $\prod_{j=1}^{\ell} f_j^{s_j}$ with a random value $U \leftarrow\!\!\$\ \mathbb{G}$. This gives the distribution:

$$(C = (f_1, \ldots, f_\ell, U \cdot m_b), C' = (\mathsf{Enc}'(\mathsf{pk}, 1), \ldots, \mathsf{Enc}'(\mathsf{pk}, \ell)) : U, f_1, \ldots, f_\ell \leftarrow\!\!\$\ \mathbb{G})$$

Claim 9.16. Hybrids D_3 and D_4 are statistically indistinguishable .

Proof. We can represent $f_i \leftarrow\!\!\$\ \mathbb{G}$ as g^{r_i} for random $r_i \leftarrow\!\!\$\ \mathbb{Z}_p$. Therefore, the term $\prod_{j=1}^{\ell} f_j^{s_j} = g^{\langle \boldsymbol{r}, \boldsymbol{s} \rangle}$ where $\boldsymbol{r} = (r_1, \ldots, r_\ell)$. Now, note that distinguishing hybrids D_3 and D_4 is at least as hard as distinguishing the following two ensembles:

$$(r_1, \ldots, r_\ell, \langle \boldsymbol{r}, \boldsymbol{s} \rangle, C' : r_1, \ldots, r_\ell \leftarrow\!\!\$\ \mathbb{Z}_p); (r_1, \ldots, r_\ell, u, C' : u, r_1, \ldots, r_\ell \leftarrow\!\!\$\ \mathbb{Z}_p)$$

If one could distinguish the second pair of distributions, then they can efficiently calculate the value of g^{r_i} and $g^{\langle \boldsymbol{r}, \boldsymbol{s} \rangle}$, thereby distinguishing the original pair of distributions.

We will now complete the proof by showing that the second pair of distributions are statistically indistinguishable. To this end, we will use LHL, as defined in Theorem 9.1. We have that

$$\mathrm{H}_\infty(\boldsymbol{s} | C', L(\boldsymbol{s}), \mathsf{pk}) = \mathrm{H}_\infty(\boldsymbol{s} | L(\boldsymbol{s}), \mathsf{pk}) \geq \mathrm{H}_\infty(\boldsymbol{s} | L(\boldsymbol{s})) - \log p \geq \ell - \lambda - \log p.$$

This is because C' is independent of the sk conditioned on pk, the value pk's component of h comes from a domain of size p, and L was a leakage function that satisfied $\mathrm{H}_\infty(\boldsymbol{s} | L(\boldsymbol{s})) = \ell - \lambda$. Now, consider, the hash function family $\mathcal{H}$ consisting of $h_{\boldsymbol{r}}(\boldsymbol{s}) = \langle \boldsymbol{r}, \boldsymbol{s} \rangle \mod p$. The output length is $m = \log p$. This is a universal hash family. To apply LHL we need, $m = k - 2\log(1/\varepsilon)$. Here $k = \ell - \lambda - \log p$. Therefore, $\log p = \ell - \lambda - \log p - 2\log(1/\varepsilon)$. Or if $\lambda \leq \ell - 2\log p - 2\log(1/\varepsilon)$, for some negligible ε then the latter two distributions are statistically indistinguishable. □

It follows from the above claim that $\mathcal{A}$ has a negligible advantage in distinguishing hybrids D_3 and D_4. Further, in Hybrid 4, the message is masked by a random value and therefore $\mathcal{A}$ has no advantage in Hybrid D_4.

Combining the different hybrid arguments together, we get that any PPT algorithm $\mathcal{A}$ has a negligible advantage in the CS+LR security game. □

Protocol DDH-Based UPKE

U-PKEG(1^κ)

Sample $\boldsymbol{s} = (s_1, \ldots, s_\ell) \leftarrow\!\!\$\ \{0,1\}$ and $g_1, \ldots, g_\ell \leftarrow\!\!\$\ \mathbb{G}$.
Compute $h = \prod_{i=1}^{\ell} g_i^{s_i}$.
return $\mathsf{sk} = \boldsymbol{s} \in \mathbb{Z}_p^\ell$, $\mathsf{pk} = (g_1, \ldots, g_\ell, h) \in \mathbb{G}^{\ell+1}$.

U-Enc(pk, $m \in \mathbb{G}$)

Parse $\mathsf{pk} = (g_1, \ldots, g_\ell, h)$
Sample $r \leftarrow\!\!\$\ \mathbb{Z}_p$
for $i = 1, \ldots, \ell$ **do**
  Compute $f_i = g_i^r$
return $C = (f_1, \ldots, f_\ell, c = h^r \cdot m) \in \mathbb{G}^{\ell+1}$

U-Dec(sk, C)

Parse $C = (f_1, \ldots, f_\ell, c = h^r \cdot m)$ and $\mathsf{sk} = \boldsymbol{s} = (s_1, \ldots, s_\ell) \in \mathbb{Z}_p^\ell$
Compute $m' = c \cdot \left(\prod_{i=1}^{\ell} f_i^{s_i}\right)^{-1}$
return m'

Upd-Pk(pk)

Parse $\mathsf{pk} = (g_1, \ldots, g_\ell, h)$
Sample $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_m) \leftarrow\!\!\$\ \{0,1\}^\ell$
Compute $h' = h \cdot \left(\prod_{i=1}^{\ell} g_i^{\delta_i}\right)$
Encrypt δ bit-by-bit, i.e., $\mathsf{up} = \left(\mathsf{U\text{-}Enc}(\mathsf{pk}, g^{\delta_1}), \ldots, \mathsf{U\text{-}Enc}(\mathsf{pk}, g^{\delta_\ell})\right)$.
return $(\mathsf{up}, \mathsf{pk}' = (g_1, \ldots, g_\ell, h'))$

Upd-Sk(sk, up)

Parse $\mathsf{up} = (c_1, \ldots, c_\ell)$
for $i = 1, \ldots, \ell$ **do**
  Compute $u_i = \mathsf{U\text{-}Dec}(\mathsf{sk}, c_i)$
  if $u_i = 1$ **then**
    Set $\delta_i = 0$
  else
    Set $\delta_i = 1$
Compute $\boldsymbol{s}' = \boldsymbol{s} + \boldsymbol{\delta}$ where $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_\ell)$ and addition is element-by-element over $\mathbb{Z}_p$.
return $\mathsf{sk}' = (\boldsymbol{s}')$

Construction 9.2: DDH Based Construction. Let κ be the the security parameter. Let $\mathcal{G}$ be a probabilistic polynomial-time "group generator" that takes as input 1^κ and outputs the description of a group $\mathbb{G}$ with prime order $p = p(\kappa)$ and g is a fixed generator of $\mathbb{G}$. Set $\ell = \lceil 5 \log p \rceil$.

9.4.3 UPKE Construction

In this section, we present our construction of an updatable public key encryption based on the BHHO Cryptosystem. This is presented in Construction 9.2.

Correctness. Informally, correctness requires that any message m encrypted by an updated public key decrypts with the help of the corresponding updated secret key to the same message m, always.

- Let $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{U\text{-}PKEG}(1^\kappa)$. Here, $\mathsf{sk} = \boldsymbol{s} = (s_1, \ldots, s_\ell) \leftarrow\!\!\$\ \{0,1\}^\ell$,and $\mathsf{sk} = (g_1, \ldots, g_\ell, \prod_{j=1}^{\ell} g_i^{s_i})$.
- Let r be the randomness used for the Upd-Sk procedure. Let $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_\ell)$ be the first ℓ bits of r. We have $(\mathsf{pk}', \mathsf{up}) \leftarrow \mathsf{Upd\text{-}Pk}(\mathsf{pk})$. Here, $\mathsf{pk}' = (g_1, \ldots, g_\ell, h \cdot \prod_{j=1}^{\ell} g_i^{\delta_i})$. $\mathsf{up} = (g_1^{r_1}, \ldots, g_\ell^{r_1}, h^{r_1} \cdot g^{\delta_1}), \ldots, (g_1^{r_\ell}, \ldots, g_\ell^{r_\ell}, h^{r_\ell} \cdot g^{\delta_\ell})$.

- We will look at Upd-Sk now. It is easy to verify that Upd-Sk correctly decrypts each ciphertext in up to corresponding g^{δ_i}. This is either 1 when $\delta_i = 0$ or non-identity if $\delta = 1$. It then updates $\boldsymbol{s}' = \boldsymbol{s} + \boldsymbol{\delta}$. Interestingly, while $\boldsymbol{s}$ was initialized to be a bit string, each element grows slowly over $\mathbb{Z}_p$.

- Consider U-Enc(pk', m). The resulting ciphertext is $(g_1^u, \ldots, g_\ell^u, h'^u \cdot m)$ for $u \leftarrow\!\!\$\, \mathbb{Z}_p$.

- Consider U-Dec$(\mathsf{sk}', g_1^u, \ldots, g_\ell^u, h'^u \cdot m))$. The decryption algorithm returns:

$$\begin{aligned} h'^u \cdot m \cdot (\prod_{j=1}^{\ell} (g_j^u)^{s'_j})^{-1} &= (h \cdot \prod_{j=1}^{\ell} g_j^{\delta_j})^u \cdot m \cdot (\prod_{j=1}^{\ell} (g_j^u)^{s'_j})^{-1} \\ &= (\prod_{j=1}^{\ell} g_j^{s_j} \cdot \prod_{j=1}^{\ell} g_j^{\delta_j})^u \cdot m \cdot (\prod_{j=1}^{\ell} (g_j^u)^{s_j + \delta_j})^{-1} \\ &= m \end{aligned}$$

- The same can be extended to additional updates. The key point to note is that the algorithms do not need $\boldsymbol{s}$ to be a bit string and therefore can, and indeed will grow.

9.4.4 Security of the UPKE Construction

Theorem 9.17. *Under the DDH Assumption, Construction 9.2 is IND-CR-CPA secure UPKE.*

Proof. We proved in Theorem 9.10 that Construction 9.1 is CS+LR secure with $\lambda = \ell - 2\log p - \omega(\log \kappa)$, under the DDH Assumption. We will use this as the starting point and use an adversary $\mathcal{A}$ against the IND-CPA game of the UPKE construction to construct an adversary $\mathcal{B}$ against the CS+LR Security game of the PKE Scheme.

- The reduction $\mathcal{B}$ receives from the challenger the public key pk_0 corresponding to some secret key $\boldsymbol{s}_0$.

- It has a time period counter t initialized to 0

- $\mathcal{B}$ provides pk_0 to the adversary $\mathcal{A}$.
- $\mathcal{B}$ responds as follows to the oracle queries to $\mathcal{O}_{upd}(\cdot)$ as follows:

  For each input invocation, it increments the counter t to i and records the $\boldsymbol{\delta}_i$ it receives as input.
- $\mathcal{B}$ then receives the challenge messages m_0^*, m_1^*.
- $\mathcal{B}$ then provides the *randomized* leakage function $L(\mathsf{sk}; \boldsymbol{\delta}^*) = \boldsymbol{s}_0 + \boldsymbol{\delta}^*$ where the addition is element-by-element over $\mathbb{Z}_p$. Looking ahead, $\boldsymbol{\delta}^*$ will correspond to the randomness for the fresh update before the secret key is provided to the $\mathcal{A}$. It also sets m_0^*, m_1^* as its challenge messages.
- $\mathcal{B}$ sends to its challenger the leakage function L, m_0^*, m_1^*. It also specifies the function f to be the encryption of each bit of the secret key in the exponent.
- In response, $\mathcal{B}$ receives C which is an encryption of m_b^* under pk_0, C' which is a encryption of $\boldsymbol{s}_0$, bit-by-bit in the exponent, under pk_0, and a leakage $\boldsymbol{z}$ on $\boldsymbol{s}_0$ defined by $\boldsymbol{z} = \boldsymbol{s}_0 + \boldsymbol{\delta}^*$ for *unknown* $\boldsymbol{\delta}^* \leftarrow\!\!\$\ \{0,1\}^\ell$. More formally,

$$C = \mathsf{U\text{-}Enc}(\mathsf{pk}_0, m_b^*); C' = \left(\mathsf{U\text{-}Enc}(\mathsf{pk}_0, g^{s_1}), \ldots, \mathsf{U\text{-}Enc}(\mathsf{pk}_0, g^{s_\ell})\right)$$

- At this point, let the time period be q'. Now, $\mathcal{A}$ expects $c^* = \mathsf{U\text{-}Enc}(\mathsf{pk}_{q'}, m_b^*)$. So $\mathcal{B}$ does the following to compute c^*:
  - $\mathcal{B}$ has $C = \left(\mathsf{U\text{-}Enc}(\mathsf{pk}_0, m_b^*)\right)$ or $C = (f_1, \ldots, f_\ell, c = h^r \cdot m_b^*)$.
  - It computes $\boldsymbol{\Delta}' = \sum_{i=1}^{q'} \boldsymbol{\delta}_i$. $\boldsymbol{\Delta} = (\Delta'_1, \ldots, \Delta'_\ell)$

– To convert it into a public key corresponding to $\boldsymbol{s}_{q'} = \boldsymbol{s}_0 + \boldsymbol{\Delta}'$, we do the following:

$$c^* = \left(f_1, \ldots, f_\ell, c \cdot \prod_{j=1}^{\ell} f_j^{\Delta'_j} \right)$$

This is where we employ the key homomorphism property.

- $\mathcal{B}$ sends to $\mathcal{A}$ the value of c^*.
- $\mathcal{B}$ continues to respond to $O_{upd}(\cdot)$ queries as before. When $\mathcal{A}$ finally stops, let q be the time period. Now, $\mathcal{B}$ does the following:
  - To compute $\boldsymbol{s}^*$:
    * It sets $\boldsymbol{\Delta} = \sum_{i=1}^{q} \boldsymbol{\delta}_i$. Again, the operation is element-by-element addition over $\mathbb{Z}_p$. Let $\boldsymbol{\Delta} = (\Delta_1, \ldots, \Delta_\ell)$.
    * With the knowledge of $\boldsymbol{z}$ and $\boldsymbol{\Delta}$, $\mathcal{B}$ sets $\boldsymbol{s}^* = \boldsymbol{s}_{q+1} = \boldsymbol{z} + \boldsymbol{\Delta}$. Recall that $\boldsymbol{z} = \boldsymbol{s}_0 + \boldsymbol{\delta}^*$ for random $\boldsymbol{\delta}^*$. In other words, $\mathcal{B}$ implicitly sets $\boldsymbol{\delta}_{q+1} = \boldsymbol{\delta}^*$, corresponding to the final secure update.
  - To compute pk^*: With the knowledge of $\boldsymbol{s}^*$ it is also easy to generate the corresponding public key pk^* by merely computing the value of $h^* = \prod_{i=1}^{\ell} g_i^{s_i^*}$ where $\boldsymbol{s}^* = (s_1^*, \ldots, s_\ell^*)$. Therefore,
  $$\mathsf{pk}^* = (g_1, \ldots, g_\ell, h^*)$$
  - To compute up^*:
    * Recall that $\mathsf{up}^* = \left(\mathsf{U\text{-}Enc}(\mathsf{pk}_q, g^{\delta_1}), \ldots \mathsf{U\text{-}Enc}(\mathsf{pk}_q, g^{\delta_\ell} \right)$ where $\boldsymbol{\delta}^* = (\delta_1, \ldots, \delta_\ell)$).
    * $\mathcal{B}$ has $C' = \left(\mathsf{U\text{-}Enc}(\mathsf{pk}_0, g^{s_1}), \ldots, \mathsf{U\text{-}Enc}(\mathsf{pk}_0, g^{s_\ell}) \right)$ where $\boldsymbol{s}_0 = (s_1, \ldots, s_\ell))$
    * Note that for all $i = 1, \ldots, \ell$, by definition, $\delta_i = z_i - s_i \in \mathbb{Z}_p$.
    * Let $\boldsymbol{ct}_i = \mathsf{Enc}(\mathsf{pk}_0, g^{s_i}) = (f_1, \ldots, f_\ell, c = h^r \cdot g^{s_i})$

* For $i = 1, \ldots \ell$, then we transform each $\boldsymbol{ct}_i$ into $\boldsymbol{ct}'_i$ where

$$\boldsymbol{ct}'_i = \left(f'_1 = f_1^{-1}, \ldots, f'_\ell = f_\ell^{-1}, c' = \left(c \cdot g^{-z_i} \cdot \prod_{j=1}^{\ell} f_j^{-\Delta_j} \right)^{-1} \right)$$

Now,

$$\mathsf{up}^* = (\boldsymbol{ct}'_1, \ldots, \boldsymbol{ct}'_\ell)$$

- Send $(\mathsf{pk}^*, \mathsf{sk}^*, \mathsf{up}^*)$ to $\mathcal{A}$.
- $\mathcal{B}$ forwards $\mathcal{A}$'s guess as its own.

Analysis of Reduction. We first show that the leakage function defined here has sufficiently small entropy loss.

Claim 9.18. $\mathrm{H}_\infty(\boldsymbol{s}_0|\boldsymbol{z}) = \ell - \lambda$ where $\lambda = \ell(1 - \log_2(4/3))$

Proof. First note that the components of $\boldsymbol{z} = (z_1, \ldots, z_\ell)$ and $\boldsymbol{s}_0 = (s_1, \ldots, s_\ell)$ are independent of each other so $\mathrm{H}_\infty(\boldsymbol{s}_0|\boldsymbol{z}) = \sum_i \mathrm{H}_\infty(s_i|z_i)$. Now, the distribution of z_i is given by

$$\Pr[z_i = 0] = \Pr[z_i = 2] = 1/4, \Pr[z_i = 1] = 1/2$$

Further,

$$\Pr[s_i = 0|z_i = 0] = 1, \Pr[s_i = 0|z_i = 2] = 0, \Pr[s_i = 0|z_i = 1] = \frac{1}{2}$$

Therefore, $\mathrm{H}_\infty(s_i|z_i) = -\log_2(1 \cdot \Pr[z_i = 0] + 1 \cdot \Pr[z_i = 2] + \frac{1}{2}\Pr[z_i = 1]) = -\log(3/4)$ and $\mathrm{H}_\infty(\boldsymbol{s}_0|\boldsymbol{z}) = \ell \cdot \log_2(4/3)$. □

We now show that the distribution of ciphertext is correct. We will show it is correct for any i. We have $\boldsymbol{c}_i = (f_1, \ldots, f_\ell, c = h^r \cdot g^{s_i})$. First, we first transform it into a cipher text of $\boldsymbol{z} - \boldsymbol{s}_0$, under pk_0. This is message homomorphism. We then transform this ciphertext, under pk_0 to a ciphertext encrypting the same message under pk_q. This is the property of key homomorphism.

- Multiplying c with g^{-z_i} gives us a valid encryption of $g^{s_i - z_i}$. However, we have that $z_i - s_i = d_i$ where $\boldsymbol{\delta}^* = (d_1, \ldots, d_\ell)$.
- To obtain the encryption of $g^{z_i - s_i}$ we merely take the inverse of all elements, and then multiply the last element by g^{z_i}. Therefore,

$$\boldsymbol{c}'_i = (f'_1 = g_1^{r'} = f_1^{-1}, \ldots, f'_\ell = g_\ell^{r'} = f_\ell^{-1}, c' = c^{-1} \cdot g^{z_i} = h^{r'} \cdot g^{-s_i} \cdot g^{z_i})$$

  with $r' = -r$.
- Now, note that $\mathsf{sk}_q = \mathsf{sk}_0 + \boldsymbol{\Delta} = (s_1 + \Delta_1, \ldots, s_\ell + \Delta_\ell)$. The public key is therefore $\mathsf{pk}_q = (g_1, \ldots, g_\ell, h_q)$ where $h_q = h \cdot \prod_{j=1}^{\ell} g_j^{\Delta_j}$. In order to transform a ciphertext $c'_i = (f'_1, \ldots, f'_\ell, c')$ under pk_0 to a ciphertext under pk_q we modify the last component, c' as $c' \cdot \prod_{j=1}^{\ell} f_j'^{\Delta_j} = (c \cdot g^{-z_i} \cdot \prod_{j=1}^{\ell} f_j^{\Delta_j})^{-1}$.

Under this reduction, it is easy to see that $\mathcal{B}$ perfectly simulates the IND-CR-CPA game for $\mathcal{A}$. The advantage of $\mathcal{A}$ against the IND-CR-CPA is the same as the advantage of $\mathcal{B}$. □

Choice of Parameters. We have from Theorem 9.10 that $\lambda \leq \ell - 2\log p - \omega(\log \kappa)$. We have also shown that our reduction needs $\ell - \lambda = \ell \cdot \log_2(4/3)$. Therefore, we have that $\ell \geq \frac{2}{\log_2(4/3)} \log p + \omega(\log \kappa)$. Or, $\ell = \lceil 5 \log p \rceil$.

9.5 Constructions based on LWE

This section presents construction from the LWE Assumption. We begin by presenting a slightly modified version of the dual-Regev PKE Scheme [Reg05, GPV08] in section 9.5.1. We show that the scheme is CS+LR secure in section 9.5.2. We then present our construction of a UPKE scheme (section 9.5.3), extended from the PKE scheme. We finally prove that the UPKE scheme is IND-CR-CPA secure in section 9.5.4.

Protocol Dual Regev or GPV Cryptosystem

$\underline{\mathsf{Gen}(1^\kappa)}$

Sample $\boldsymbol{A} \leftarrow_s \mathbb{Z}_p^{n \times m}$
Sample $r \leftarrow_s \{0,1\}^m$
Compute $\boldsymbol{u} = \boldsymbol{A}\boldsymbol{r}$
return $(\mathsf{pk} = (\boldsymbol{A}, \boldsymbol{u}), \mathsf{sk} = (\boldsymbol{r}))$

$\underline{\mathsf{Enc}(\mathsf{pk}, b \in \{0,1\})}$

Parse $\mathsf{pk} = (\boldsymbol{A}, \boldsymbol{u})$
Sample $\boldsymbol{x} \leftarrow_s \mathbb{Z}_p^n, \boldsymbol{e} \leftarrow_s \chi^m, e' \leftarrow \chi'$
Compute $\boldsymbol{t} = \boldsymbol{A}^T\boldsymbol{x} + \boldsymbol{e}$, $pad = \langle \boldsymbol{x}, \boldsymbol{u} \rangle + e' + b\lfloor p/2 \rfloor$
return $c = (\boldsymbol{t}, pad)$

$\underline{\mathsf{Dec}(\mathsf{sk}, c)}$

Parse $c = (\boldsymbol{t}, pad)$ and $\mathsf{sk} = \boldsymbol{r}$
Compute $b' = (pad - \langle \boldsymbol{r}, \boldsymbol{t} \rangle) \in \mathbb{Z}_p$
return 0 if m' is closer to 0 than to $\lfloor p/2 \rfloor$ and 1 otherwise.

Construction 9.3: The Dual Regev or GPV Cryptosystem. Let n, m, p be integer parameters of the scheme. We will assume that LWE holds where p is super-polynomial and χ is polynomially bounded. Then, we set χ' to be uniformly random over (say) $[-p/8, p/8]$.

9.5.1 The Dual Regev or GPV Cryptosystem

The construction is presented as Construction 9.3.

Correctness. We show that the decryption algorithm is correct with overwhelming probability (over the choice of the randomness of Gen, Enc). The decryption algorithm computes:

$$\langle \boldsymbol{r}, \boldsymbol{t} \rangle = \langle \boldsymbol{r}, \boldsymbol{A}^T\boldsymbol{x} + \boldsymbol{e} \rangle = \langle \boldsymbol{r}, \boldsymbol{A}^T\boldsymbol{x} \rangle + \langle \boldsymbol{r}, \boldsymbol{e} \rangle = \langle \boldsymbol{x}, \boldsymbol{u} \rangle + \langle \boldsymbol{r}, \boldsymbol{e} \rangle$$

$$pad - \langle \boldsymbol{r}, \boldsymbol{t} \rangle = \langle \boldsymbol{x}, \boldsymbol{u} \rangle + e' + b\left\lfloor \frac{p}{2} \right\rfloor - \langle \boldsymbol{r}, \boldsymbol{t} \rangle = b\left\lfloor \frac{p}{2} \right\rfloor + (e' - \langle \boldsymbol{e}, \boldsymbol{r} \rangle)$$

Now, note that $e' - \langle e, r \rangle$ is small in comparison to p. Therefore, the computed value is closer to $\lfloor p/2 \rfloor$ when $b = 0$ and the opposite when $b = 1$.

9.5.2 CS+LR Security of the dual-Regev Cryptosystem

In this section, we provide proof of the combined circular security and leakage resilience of the dual-Regev Cryptosystem. Formally, we will prove the following theorem:

Theorem 9.19. *Under the LWE Assumption, Construction 9.3 is λ-CS+LR secure with leakage $\lambda = m - (n+1)\log p - \omega(\log \kappa)$.*

Before we can prove the above theorem, we show the existence of an encryption algorithm Enc' such that

$$(\mathsf{Enc}'(\mathsf{pk}, i), \mathsf{sk}) \approx_c (\mathsf{Enc}(\mathsf{pk}, r_i), \mathsf{sk}).$$

Consider: $\mathsf{Enc}'(\mathsf{pk}, i) := (\boldsymbol{t}', pad')$ where:

- Let $\boldsymbol{x} \leftarrow\!\!\$\ \mathbb{Z}_p^n, \boldsymbol{e} \leftarrow\!\!\$\ \chi^m$ and $\boldsymbol{d} = (d_1 = 0, \ldots, d_{i-1} = 0, d_i = -\lfloor p/2 \rfloor, d_{i+1} = 0, \ldots, d_m = 0)$. Then, $\boldsymbol{t}' = \boldsymbol{A}^T\boldsymbol{x} + \boldsymbol{e} + \boldsymbol{d}$.
- $pad' = \langle \boldsymbol{x}, \boldsymbol{u} \rangle + e'$ where e' is chosen from a distribution χ' such that $e' + B$ is statistically indistinguishable from e' where $B \in \mathbb{Z}_p$.

Lemma 9.20. *Under the LWE Assumption,* $(\mathsf{pk}, \mathsf{Enc}'(\mathsf{pk}, i), \mathsf{sk}) \approx_c (\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, r_i), \mathsf{sk})$. *where* $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{Gen}(1^\kappa)$

Proof. We will prove through a sequence of hybrids. This is a tabulated summary of the changes through the proof:

Hybrid	Hybrid Definition	Security
D_0	Enc is used to encrypt r_i	Lemma 9.2
D_1	D_0 except $\langle \boldsymbol{x}, \boldsymbol{u} \rangle$ replaced with $\langle \boldsymbol{r}, \boldsymbol{t} \rangle$	LWE
D_2	D_1 except $\boldsymbol{t} = \boldsymbol{A}^T\boldsymbol{x} + \boldsymbol{e}$ replaced with $\boldsymbol{t} \leftarrow\!\!\$\ \mathbb{Z}_p^n$	Identical
D_3	D_2 except $\boldsymbol{t}$ replaced with $\boldsymbol{t} + \boldsymbol{d}$ where $\boldsymbol{d} = (0, \ldots, 0, d_i = -\lfloor p/2 \rfloor, 0, \ldots, 0)$	LWE
D_4	D_3 except $\boldsymbol{t} = \boldsymbol{A}^T\boldsymbol{x} + \boldsymbol{e}$ where $\boldsymbol{x} \leftarrow\!\!\$\ \mathbb{Z}_p^n, \boldsymbol{e} \leftarrow\!\!\$\ \chi^m$	Lemma 9.2
D_5	Enc′ is used to encrypt r_i	

Note that each of our hybrid distribution contains the value of pk. We drop the term for convenience. pk

Hybrid D_0. It corresponds to the distribution using Enc. This is the distribution:

$$(\boldsymbol{t} = \boldsymbol{A}^T\boldsymbol{x} + \boldsymbol{e}, pad = \langle \boldsymbol{x}, \boldsymbol{u} \rangle + e' + r_i\lfloor p/2 \rfloor, \boldsymbol{r} : x \leftarrow\!\!\$\ \mathbb{Z}_p^n, \boldsymbol{e} \leftarrow\!\!\$\ \chi^m, e' \leftarrow\!\!\$\ \chi')$$

Hybrid D_1. It corresponds to the distribution:

$$(\boldsymbol{t} = \boldsymbol{A}^T\boldsymbol{x} + \boldsymbol{e}, pad' = \langle \boldsymbol{r}, \boldsymbol{t} \rangle + e' + r_i\lfloor p/2 \rfloor, \boldsymbol{r} : \boldsymbol{x} \leftarrow\!\!\$\ \mathbb{Z}_p^n, \boldsymbol{e} \leftarrow\!\!\$\ \chi^m, e' \leftarrow\!\!\$\ \chi')$$

Hybrids D_0 and D_1 are statistically indistinguishable from each other because e' is statistically indistinguishable from $e' + \langle \boldsymbol{e}, \boldsymbol{r} \rangle$. This follows from Lemma 9.2. Therefore, an adversary $\mathcal{A}$ has negligible advantage in distinguishing between Hybrids D_0 and D_1.

Hybrid D_2. It corresponds to the distribution where the first term is actually a random vector chosen from $\mathbb{Z}_p^m$.

$$(\boldsymbol{t}, pad' = \langle \boldsymbol{r}, \boldsymbol{t} \rangle + e' + r_i \lfloor p/2 \rfloor, \boldsymbol{r} : \boldsymbol{t} \leftarrow_\$ \mathbb{Z}_p^n, \boldsymbol{e} \leftarrow_\$ \chi^m, e' \leftarrow_\$ \chi')$$

Claim 9.21. Under the LWE Assumption, the advantage of any PPT attacker $\mathcal{A}$ in distinguishing between Hybrids D_1 and D_2 is negligible.

Proof. We will use an adversary $\mathcal{A}$ capable of distinguishing between Hybrids D_1, D_2 to create an adversary $\mathcal{B}$ that can win against the LWE game. After receiving input from the challenger $(\boldsymbol{A}, \boldsymbol{y})$, the adversary runs Gen to get $(\mathsf{pk}, \mathsf{sk})$. It chooses $e' \leftarrow_\$ \chi'$ and sends to $\mathcal{A}$: $(\boldsymbol{y}, \langle \boldsymbol{y}, \boldsymbol{r} \rangle + e' + r_i \lfloor p/2 \rfloor, \mathsf{sk})$. It is easy to see that $\mathcal{B}$ perfectly simulates one of the hybrids based on the input it receives. It forwards $\mathcal{A}$'s guess (guessing 0 for Hybrid D_1 and 1 for Hybrid D_2) as its own. This concludes the proof that $\mathcal{A}$ has a negligible advantage in distinguishing between the two hybrids. □

Hybrid D_3. Same as hybrid D_2 except that we have added $\lfloor p/2 \rfloor$ to element i of $\boldsymbol{t}$. Or the first term is $\boldsymbol{t} + \boldsymbol{d}$ where $\boldsymbol{d} = (d_1 = 0, \ldots, d_{i-1} = 0, d_i = -\lfloor p/2 \rfloor, d_{i+1} = 0, \ldots, d_m = 0)$

$$(\boldsymbol{t}' = \boldsymbol{t} + \boldsymbol{d}, pad' = \langle \boldsymbol{r}, \boldsymbol{t}' \rangle + e' + r_i \lfloor p/2 \rfloor, \boldsymbol{r} : \boldsymbol{t} \leftarrow_\$ \mathbb{Z}_p^n, \boldsymbol{e} \leftarrow_\$ \chi^m, e' \leftarrow_\$ \chi')$$

We know that $\boldsymbol{t}' = \boldsymbol{t} + \boldsymbol{d}$ for a fixed $\boldsymbol{d}$ is indistinguishable from $\boldsymbol{t}$ where $\boldsymbol{t} \leftarrow_\$ \mathbb{Z}_p^n$. Therefore, $\mathcal{A}$ has no advantage in distinguishing between Hybrids D_2 and D_3.

Hybrid D_4. Let $\boldsymbol{d} = (d_1 = 0, \ldots, d_{i-1} = 0, d_i = -\lfloor p/2 \rfloor, d_{i+1} = 0, \ldots, d_m = 0)$. Then, we have the distribution:

$$(\boldsymbol{t} = \boldsymbol{A}^T \boldsymbol{x} + \boldsymbol{e} + \boldsymbol{d}, pad' = \langle \boldsymbol{r}, \boldsymbol{t} \rangle + e' + r_i \lfloor p/2 \rfloor, \boldsymbol{r} : \boldsymbol{x} \leftarrow_\$ \mathbb{Z}_p^n, \boldsymbol{e} \leftarrow_\$ \chi^m, e' \leftarrow_\$ \chi')$$

Claim 9.22. Under the LWE Assumption, the advantage of any PPT attacker $\mathcal{A}$ in distinguishing between Hybrids D_3 and D_4 is negligible.

The proof is similar to the earlier claim.

Hybrid D_5. We have the distribution corresponding to Enc′, i.e.,

$$(\boldsymbol{t} = \boldsymbol{A}^T\boldsymbol{x} + \boldsymbol{e} + \boldsymbol{d}, pad' = \langle \boldsymbol{x}, \boldsymbol{u} \rangle + e', \boldsymbol{r} : \boldsymbol{x} \leftarrow\!\!\$\ \mathbb{Z}_p^n, \boldsymbol{e} \leftarrow\!\!\$\ \chi^m, e' \leftarrow\!\!\$\ \chi')$$

Let us simplify the pad' term in Hybrid D_4.

$$\begin{aligned} pad' &= \langle \boldsymbol{r}, \boldsymbol{t} \rangle + e' + r_i \lfloor p/2 \rfloor \\ &= \langle \boldsymbol{r}, \boldsymbol{A}^T\boldsymbol{x} + \boldsymbol{e} + \boldsymbol{d} \rangle + e' + r_i \lfloor p/2 \rfloor \\ &= \langle \boldsymbol{r}, \boldsymbol{A}^T\boldsymbol{x} \rangle + \langle \boldsymbol{r}, \boldsymbol{e} \rangle + \langle \boldsymbol{r}, \boldsymbol{d} \rangle + e' + r_i \lfloor p/2 \rfloor \\ &= \langle \boldsymbol{u}, \boldsymbol{x} \rangle + e' + \langle \boldsymbol{r}, \boldsymbol{e} \rangle + r_i(-\lfloor p/2 \rfloor) + r_i(\lfloor p/2 \rfloor) \\ &= \langle \boldsymbol{u}, \boldsymbol{x} \rangle + e' + \langle \boldsymbol{r}, \boldsymbol{e} \rangle \end{aligned}$$

We again use Lemma 9.2 to show that pad' in Hybrid D_4 is statistically indistinguishable from pad' in Hybrid D_5. Therefore, an adversary $\mathcal{A}$ has a negligible advantage in distinguishing hybrids D_4 and D_5.

Therefore, we have shown that $(\mathsf{Enc}(\mathsf{sk}, r_i), \mathsf{sk}) \approx_c (\mathsf{Enc}'(\mathsf{pk}, i), \mathsf{sk})$ □

Further, note that $\boldsymbol{r}$ is independently chosen, bit by bit. In addition, each Enc, Enc′ has independently chosen randomness. Therefore, as a corollary we get that:

Corollary 9.23. *Under the LWE Assumption,*

$$(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, r_1), \ldots, \mathsf{Enc}(\mathsf{pk}, r_m), \mathsf{sk}) \approx_c (\mathsf{pk}, \mathsf{Enc}'(\mathsf{pk}, 1), \ldots, \mathsf{Enc}'(\mathsf{pk}, m), \mathsf{sk})$$

We can now prove the original theorem:

Theorem 9.19. *Under the LWE Assumption, Construction 9.3 is λ-CS+LR secure with leakage $\lambda = m - (n+1)\log p - \omega(\log \kappa)$.*

Proof. We prove this similar to the proof of Theorem 9.10. This is done through a sequence of hybrids. This is a tabulated summary of the hybrids changes through the proof:

Hybrid	Hybrid Definition	Security
D_0	The Original CS+LR Security Game, Enc is used	Corollary 9.23
D_1	D_0 except Enc′ is used	Identical
D_2	D_1 except $\langle \boldsymbol{x}, \boldsymbol{u} \rangle$ replaced with $\langle \boldsymbol{r}, \boldsymbol{t} \rangle$	LWE
D_3	D_2 except $\boldsymbol{t} = \boldsymbol{A}^T\boldsymbol{x} + \boldsymbol{e}$ replaced with $\boldsymbol{t} \leftarrow\!\!\$\ \mathbb{Z}_p^n$.	Leftover Hash Lemma
D_4	D_3 except $\langle \boldsymbol{r}, \boldsymbol{t} \rangle$ replaced with $U \leftarrow \mathbb{Z}_p$	

Note that each of our hybrid distribution contains pk and $L(\mathsf{sk} = \boldsymbol{r})$ in its definition where $\boldsymbol{r} \leftarrow\!\!\$\ \{0,1\}^m$. We drop these terms from the definition for simplicity and merely focus on the two ciphertexts which undergo the bulk of the changes.

Hybrid D_0. The original CS+LR Security Game. We get the distribution:

$$\Big(C = (\boldsymbol{t} = \boldsymbol{A}^T\boldsymbol{x} + \boldsymbol{e}, pad = \langle \boldsymbol{u}, \boldsymbol{x} \rangle + e' + b\lfloor p/2 \rfloor), C' = (\mathsf{Enc}(\mathsf{pk}, r_1), \ldots, \mathsf{Enc}(\mathsf{pk}, r_m)) \\ : \boldsymbol{x} \leftarrow\!\!\$\ \mathbb{Z}_p^n, \boldsymbol{e} \leftarrow\!\!\$\ \chi^m, e' \leftarrow\!\!\$\ \chi'\Big).$$

Hybrid D_1. This is the same game as Hybrid D_0, except C' is generated using the "fake encryption" algorithm.

$$\left(C = (\boldsymbol{t}, pad, C' = (\mathsf{Enc}'(\mathsf{pk}, 1), \ldots, \mathsf{Enc}'(\mathsf{pk}, m)) : \boldsymbol{x} \leftarrow\!\!\$\ \mathbb{Z}_p^n, \boldsymbol{e} \leftarrow\!\!\$\ \chi^m, e' \leftarrow\!\!\$\ \chi'\right)$$

From Corollary 9.23 we have that $\mathcal{A}$ has a negligible advantage in distinguishing the two distributions. This is valid because we showed that the two distributions are computationally indistinguishable even under total leakage of sk. Here, we only have partial leakage $L(\mathsf{sk})$.

Hybrid D_2. This is the same game as Hybrid D_1, with the difference in the second component of C (pad). This gives the distribution:

$$\left(C = (\boldsymbol{t}, pad' = \langle \boldsymbol{r}, \boldsymbol{t} \rangle + e' + b\lfloor p/2 \rfloor), C') : \boldsymbol{x} \leftarrow\!\!\$\ \mathbb{Z}_p^n, \boldsymbol{e} \leftarrow\!\!\$\ \chi^m, e' \leftarrow\!\!\$\ \chi'\right)$$

Hybrids D_1 and D_2 are statistically indistinguishable because e' is statistically indistinguishable from $e' + \langle \boldsymbol{e}, \boldsymbol{r} \rangle$ The LWE challenge is embedded in the challenge ciphertext.

Hybrid D_3. We replace computed $\boldsymbol{t}$ with a randomly chosen vector $\boldsymbol{t}$ in Hybrid D_2.

$$(\mathsf{pk}, L(\mathsf{sk}), C = (\boldsymbol{t}, pad' = \langle \boldsymbol{r}, \boldsymbol{t} \rangle + e' + b\lfloor p/2 \rfloor), C') : \boldsymbol{t} \leftarrow\!\!\$\ \mathbb{Z}_p^m, e' \leftarrow\!\!\$\ \chi')$$

Claim 9.24. Under the LWE Assumption, the advantage of any PPT attacker $\mathcal{A}$ in distinguishing between Hybrids D_2 and D_3 is negligible.

The proof is similar to that of the earlier claim in this section.

Hybrid D_4. We replace $\langle \boldsymbol{r}, \boldsymbol{t} \rangle$ with a randomly chosen element $u \leftarrow\!\!\$\ \mathbb{Z}_p$. This provides the distribution:

$$(\mathsf{pk}, L(\mathsf{sk}), C = (\boldsymbol{t}, pad'' = U + e' + b\lfloor p/2 \rfloor), C') : U \leftarrow\!\!\$\ \mathbb{Z}_p, \boldsymbol{t} \leftarrow\!\!\$\ \mathbb{Z}_p^m, e' \leftarrow\!\!\$\ \chi')$$

Claim 9.25. Hybrids D_3 and D_4 are statistically indistinguishable.

Proof. To prove the above claim, we will use LHL, as defined in Theorem 9.1. We have that

$$k = \mathrm{H}_\infty(\mathsf{sk}|C', L(\mathsf{sk}), \mathsf{pk}) = \mathrm{H}_\infty(\mathsf{sk}|L(\mathsf{sk}), \mathsf{pk}) \geq \mathrm{H}_\infty(\mathsf{sk}|L(\mathsf{sk})) - n \log p$$

$$k \geq m - \lambda - n \log p.$$

This is because C' is independent of the sk conditioned on pk, the public key pk comes from a domain $\mathbb{Z}_p^n$ of size p^n, and L is a leakage function that satisfies $\mathrm{H}_\infty(\mathsf{sk}|L(\mathsf{sk})) = m - \lambda$.

Now, consider, the hash function family $\mathcal{H}$ consisting of $h_v(\boldsymbol{r}) = \langle \boldsymbol{v}, \boldsymbol{r} \rangle \mod p$. The output length is $\log p$ This is a universal hash family. To apply LHL we need, $\log p \leq k - 2\log(1/\varepsilon) = m - \lambda - n \log p - 2\log(1/\varepsilon)$. Therefore, if $\lambda \leq m - (n+1)\log p - 2\log(1/\varepsilon)$ for some negligible ε then the latter two distributions are statistically indistinguishable. □

It follows from the above claim that $\mathcal{A}$ has a negligible advantage in distinguishing hybrids D_3 and D_4. Further, in Hybrid D_4, the message is masked by a random value and therefore $\mathcal{A}$ has no advantage in Hybrid D_4.

Combining the different hybrid arguments together, we get that any PPT algorithm $\mathcal{A}$ has a negligible advantage in the CS+LR security game.

□

Protocol LWE-Based UPKE

U-PKEG(1^κ)

Sample $\boldsymbol{A} \leftarrow_\$ \mathbb{Z}_p^{n \times m}$
Sample $\boldsymbol{r} \leftarrow_\$ \{0,1\}^m$
Compute $\boldsymbol{u} = \boldsymbol{A}\boldsymbol{r}$
return $(\mathsf{pk} = (\boldsymbol{A}, \boldsymbol{u}), \mathsf{sk} = (\boldsymbol{r}))$

U-Enc($\mathsf{pk}, b \in \{0,1\}$)

Parse $\mathsf{pk} = (\boldsymbol{A}, \boldsymbol{u})$
Sample $\boldsymbol{x} \leftarrow_\$ \mathbb{Z}_p^n, \boldsymbol{e} \leftarrow_\$ \chi^m, e' \leftarrow \chi'$
Compute $\boldsymbol{t} = \boldsymbol{A}^T\boldsymbol{x} + \boldsymbol{e}$, $pad = \langle \boldsymbol{x}, \boldsymbol{u} \rangle + e' + b\lfloor p/2 \rfloor$
return $c = (\boldsymbol{t}, pad)$

U-Dec(sk, c)

Parse $c = (\boldsymbol{t}, pad)$ and $\mathsf{sk} = \boldsymbol{r}$
Compute $b' = (pad - \langle \boldsymbol{r}, \boldsymbol{t} \rangle) \in \mathbb{Z}_p$
return 0 if m' is closer to 0 than to $\lfloor p/2 \rfloor$ and 1 otherwise.

Upd-Pk(pk)

Parse $\mathsf{pk} = (\boldsymbol{A}, \boldsymbol{u})$
Sample $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_m) \leftarrow_\$ \{0,1\}^m$
Compute $\boldsymbol{u}' = \boldsymbol{u} + \boldsymbol{A}\boldsymbol{\delta}$
Encrypt δ bit-by-bit, i.e., $\mathsf{up} = (\text{U-Enc}(\mathsf{pk}, \delta_1), \ldots, \text{U-Enc}(\mathsf{pk}, \delta_m))$.
return $(\mathsf{up}, \mathsf{pk}' = (\boldsymbol{A}, \boldsymbol{u}'))$

Upd-Sk(sk, up)

Parse $\mathsf{up} = (c_1, \ldots, c_m)$
for $i = 1, \ldots, m$ **do**
  $\delta_i = \text{U-Dec}(\mathsf{sk}, c_i)$
Compute $\boldsymbol{r}' = \boldsymbol{r} + \boldsymbol{\delta}$ where $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_m)$
return $\mathsf{sk}' = (\boldsymbol{r}')$

Construction 9.4: LWE Based Construction. Let n, m, p be integer parameters of the scheme. We will assume that LWE holds where p is super-polynomial and χ is polynomially bounded. Then, we set χ' to be uniformly random over (say) $[-p/8, p/8]$. Further, we have that $m \geq \frac{(n+1)}{\log_2(4/3)} \log p + \omega(\log \kappa)$.

9.5.3 UPKE Construction

In this section, we present our construction of an updatable public key encryption based on the dual-Regev cryptosystem. This is presented in Construction 9.4.

Correctness. The property of correctness of UPKE requires that a bit b encrypted by an updated public key decrypts to the same bit b when the corresponding updated secret key is used.

- $(\mathsf{pk} = (\boldsymbol{A}, \boldsymbol{u} = \boldsymbol{A}\boldsymbol{r}), \mathsf{sk} = \boldsymbol{r}) \leftarrow \text{U-PKEG}(1^\kappa)$
- We have the update bit $\boldsymbol{\delta} \leftarrow_\$ \{0,1\}^m$. We have the updated public key $\mathsf{pk}' = (\boldsymbol{A}, \boldsymbol{u}')$ where $\boldsymbol{u}' = \boldsymbol{u} + \boldsymbol{A}\boldsymbol{\delta}$. We also have $\mathsf{sk}' = \boldsymbol{r}' = \boldsymbol{r} + \boldsymbol{\delta}$
- Let us look at $\text{U-Enc}(\mathsf{pk}', b)$. It produces ciphertext $(\boldsymbol{t}, pad)$ where $\boldsymbol{t} = \boldsymbol{A}^T\boldsymbol{x} + \boldsymbol{e}$, $pad = \langle \boldsymbol{x}, \boldsymbol{u}' \rangle + e' + b\lfloor p/2 \rfloor$.

- Now, let us look at U-Dec$(\boldsymbol{r}', (\boldsymbol{t}, pad))$. It computes

$$\begin{aligned} pad - \langle \boldsymbol{r}', \boldsymbol{t} \rangle &= \langle \boldsymbol{x}, \boldsymbol{u}' \rangle + e' + b\lfloor p/2 \rfloor - \langle \boldsymbol{r} + \boldsymbol{\delta}, \boldsymbol{A}^T \boldsymbol{x} + \boldsymbol{e} \rangle \\ &= \langle \boldsymbol{x}, \boldsymbol{A}\boldsymbol{r} + \boldsymbol{A}\boldsymbol{\delta} \rangle + e' + b\lfloor p/2 \rfloor - \langle \boldsymbol{r} + \boldsymbol{\delta}, \boldsymbol{A}^T \boldsymbol{x} + \boldsymbol{e} \rangle \\ &= \langle \boldsymbol{x}, \boldsymbol{A}(\boldsymbol{r} + \boldsymbol{\delta}) \rangle - \langle \boldsymbol{r} + \boldsymbol{\delta}, \boldsymbol{A}^T, \boldsymbol{x} \rangle + e' - \langle \boldsymbol{e}, \boldsymbol{r} \rangle + b\lfloor p/2 \rfloor \\ &= e' - \langle \boldsymbol{e}, \boldsymbol{r} \rangle + b\lfloor p/2 \rfloor \end{aligned}$$

- Now, note that $e' - \langle e, r \rangle$ is small in comparison to p. Therefore, the computed value is closer to $\lfloor p/2 \rfloor$ when $b = 0$ and the opposite when $b = 1$.

9.5.4 Security of the UPKE Construction

Theorem 9.26. *Under the LWE Assumption, Construction 9.4 is IND-CR-CPA secure UPKE.*

Proof. The proof is very similar to the proof of Theorem 9.17. We proved in Theorem 9.19 that the PKE scheme was CS+LR secure with $\lambda \leq m - (n+1)\log p - \omega(\log \kappa)$, under the LWE assumption. We will use this to construct $\mathcal{B}$ against the CS+LR game by using $\mathcal{A}$ against the IND-CPA Game.

- The reduction $\mathcal{B}$ receives from the challenger the public key pk_0 corresponding to some secret key $\boldsymbol{s}_0$.
- It has a time period counter t initialized to 0
- $\mathcal{B}$ provides pk_0 to the adversary $\mathcal{A}$.
- $\mathcal{B}$ responds as follows to the oracle queries to $\mathcal{O}_{upd}(\cdot)$ as follows:

  For each input invocation, it increments the counter t to i and records the $\boldsymbol{\delta}_i$ it receives as input.

- $\mathcal{B}$ then receives the challenge messages m_0^*, m_1^*.

- $\mathcal{B}$ then provides the *randomized* leakage function $L(\mathsf{sk}; \boldsymbol{\delta}^*) = \boldsymbol{s}_0 + \boldsymbol{\delta}^*$ where the addition is element-by-element over $\mathbb{Z}_p$. Looking ahead, $\boldsymbol{\delta}^*$ will correspond to the randomness for the fresh update before the secret key is provided to the $\mathcal{A}$. It also sets m_0^*, m_1^* as its challenge messages.

- $\mathcal{B}$ sends to its challenger the leakage function L, m_0^*, m_1^*. It also specifies the function f to be the encryption of each bit of the secret key.

- In response, $\mathcal{B}$ receives C which is an encryption of m_b^* under pk_0, C' which is a encryption of $\boldsymbol{s}_0$, bit-by-bit, under pk_0, and a leakage $\boldsymbol{z}$ on $\boldsymbol{r}_0$ defined by $\boldsymbol{z} = \boldsymbol{r}_0 + \boldsymbol{\delta}^*$ for *unknown* $\boldsymbol{\delta}^* \leftarrow\!\!\$\ \{0,1\}^m$. More formally,

$$C = \mathsf{U\text{-}Enc}(\mathsf{pk}_0, m_b^*); C' = \left(\mathsf{U\text{-}Enc}(\mathsf{pk}_0, r_1), \ldots, \mathsf{U\text{-}Enc}(\mathsf{pk}_0, r_m)\right)$$

- At this point, let the time period be q'. Now, $\mathcal{A}$ expects $c^* = \mathsf{U\text{-}Enc}(\mathsf{pk}_{q'}, m_b^*)$. So $\mathcal{B}$ does the following to compute c^*:
  - $\mathcal{B}$ has $C = \mathsf{U\text{-}Enc}(\mathsf{pk}_0, m_b^*)$ or $C = \left(\boldsymbol{t} = \boldsymbol{A}^T\boldsymbol{x} + \boldsymbol{e}, pad = \langle \boldsymbol{x}, \boldsymbol{u}\rangle + e' + m_b^* \lfloor p/2 \rfloor\right)$.
  - It computes $\boldsymbol{\Delta}' = \sum_{i=1}^{q'} \boldsymbol{\delta}_i$.
  - It computes $pad^* = pad + \langle \boldsymbol{\Delta}', \boldsymbol{t}\rangle$ and sets $\boldsymbol{t}^* = \boldsymbol{t}$.
  - Now, $c^* = (\boldsymbol{t}^*, pad^*)$

- $\mathcal{B}$ sends to $\mathcal{A}$ the value of c^*.

- $\mathcal{B}$ continues to respond to $\mathcal{O}_{upd}(\cdot)$ queries as before. When $\mathcal{A}$ finally stops, let q be the time period. Now, $\mathcal{B}$ does the following:
  - To compute $\mathsf{sk}^* = \boldsymbol{r}^*$:
    * Set $\boldsymbol{\Delta} = \sum_{i=1}^{q} \boldsymbol{\delta}_i$.

* With the knowledge of $\boldsymbol{z}, \Delta$, $\mathcal{B}$ sets $\boldsymbol{r}^* = \boldsymbol{r}_{q+1} = \boldsymbol{z} + \Delta$.

– To compute pk^*: With the knowledge of $\boldsymbol{A}, \boldsymbol{r}^*$, $\mathcal{B}$ computes $\boldsymbol{u}^* = \boldsymbol{A}\boldsymbol{r}^*$. It sets $\mathsf{pk}^* = (\boldsymbol{A}, \boldsymbol{u}^*)$.

– To compute up^*: $\mathcal{B}$ has bit-by-bit encryption of $\boldsymbol{r}_0$. It needs to compute the bit-by-bit encryption of $\boldsymbol{\delta}^* = \boldsymbol{z} - \boldsymbol{r}_0$. For simplicity, assume that $\boldsymbol{z}$ is a trit, i.e., taking value 0, 1, 2. Let $\boldsymbol{r}_0 = (r_1, \ldots, r_m)$, $\boldsymbol{\delta}^* = (d_1, \ldots, d_m)$ and $\boldsymbol{z} = (z_1, \ldots, z_m)$. Recall that $r_i, d_i \in \{0, 1\}$ while $z_i \in \{0, 1, 2\}$.

We will first look at how to transform $\mathsf{U\text{-}Enc}(\mathsf{pk}_0, r_i)$ to $\mathsf{U\text{-}Enc}(\mathsf{pk}_0, d_i)$.

* If $z_i = 2$, then we have that $r_i = d_i = 1$. Therefore, $\mathsf{U\text{-}Enc}(\mathsf{pk}_0, r_i) = \mathsf{U\text{-}Enc}(\mathsf{pk}_0, d_i)$ and we do not need to do anything.

* Similarly if $z_i = 0$, then we have that $r_i = d_i = 0$. Once again, $\mathsf{U\text{-}Enc}(\mathsf{pk}_0, r_i) = \mathsf{U\text{-}Enc}(\mathsf{pk}_0, d_i)$ and we do not need to do anything.

* If $z_i = 1$, then we merely need $\mathsf{U\text{-}Enc}(\mathsf{pk}_0, r_i)$ to be modified to $\mathsf{U\text{-}Enc}(\mathsf{pk}_0, 1 - r_i)$. To achieve this we merely add $\lfloor p/2 \rfloor$ to the second term in the ciphertext.

To convert $\mathsf{U\text{-}Enc}(\mathsf{pk}_0, d_i)$ to $\mathsf{U\text{-}Enc}(\mathsf{pk}_q, d_i)$ we do the following:

* Note that $\mathsf{U\text{-}Enc}(\mathsf{pk}_0, d_i) = (\boldsymbol{t}_0, pad_0)$ where $\boldsymbol{t}_0 = \boldsymbol{A}^T\boldsymbol{x} + \boldsymbol{e}$ and $pad_0 = \langle \boldsymbol{x}, \boldsymbol{u}_0 \rangle + e' + b\lfloor p/2 \rfloor$. Further, $\boldsymbol{u}_q = \boldsymbol{u}_0 + \boldsymbol{A}\Delta$

* Let $t_q = t_0$, then $pad_q = pad_0 + \langle \Delta, \boldsymbol{t}_0 \rangle$. with the choice

- Send $(\mathsf{pk}^*, \mathsf{sk}^*, \mathsf{up}^*)$ to $\mathcal{A}$.

- $\mathcal{B}$ forwards $\mathcal{A}$'s guess as its own.

Analysis of the Reduction. We first show that the leakage function has sufficiently small entropy loss.

Claim 9.27. $\mathrm{H}_\infty(\boldsymbol{r}_0 | \boldsymbol{z}) = m - \lambda$, where $\lambda = m(1 - \log_2(4/3))$

The above is identical to Claim 9.18 in the proof of Theorem 9.17.

We then need to show that the distribution of ciphertext is correct. Specifically, the distribution of the update ciphertext. We have $t_0 = t_q$. We will show that pad_q is correctly distributed. By definition we have that: $pad_q = \langle \boldsymbol{x}, \boldsymbol{u}_q \rangle + e' + b\lfloor p/2 \rfloor$. Here, we compute pad_q as follows:

$$\begin{aligned} pad_q &= pad_0 + \langle \boldsymbol{\Delta}, \boldsymbol{t}_0 \rangle \\ &= \langle \boldsymbol{x}, \boldsymbol{u}_0 \rangle + e' + b\lfloor p/2 \rfloor + \langle \boldsymbol{\Delta}, \boldsymbol{t}_0 \rangle \\ &= \langle \boldsymbol{x}, \boldsymbol{u}_0 \rangle + e' + b\lfloor p/2 \rfloor + \langle \boldsymbol{\Delta}, \boldsymbol{A}^T \boldsymbol{x} + \boldsymbol{e} \rangle \\ &= \langle \boldsymbol{x}, \boldsymbol{u}_0 \rangle + \langle \boldsymbol{A}\boldsymbol{\Delta}, \boldsymbol{x} \rangle + e' + \langle \boldsymbol{\Delta}, \boldsymbol{e} \rangle + b\lfloor p/2 \rfloor \\ &= \langle \boldsymbol{x}, \boldsymbol{u}_q \rangle + e' + \langle \boldsymbol{\Delta}, \boldsymbol{e} \rangle + b\lfloor p/2 \rfloor \end{aligned}$$

We can now use the definition of distribution e' and Lemma 9.2 to show that the computed distribution is statistically indistinguishable from the actual distribution.

Under this reduction, it is easy to see that $\mathcal{B}$ perfectly simulates the IND-CPA game for $\mathcal{A}$. The advantage of $\mathcal{A}$ against the IND-CPA is the same as the advantage of $\mathcal{B}$. □

Choice of Parameters. From Theorem 9.19, we have that $m - \lambda \geq (n+1)\log p + \omega(\log \kappa)$. Further, we have from the above claim that $m - \lambda = m \log_2(4/3)$. Putting the two together, we get $m \geq \frac{n+1}{\log_2(4/3)} \log p + \omega(\log \kappa)$.

10 | Conclusion and Final Thoughts

In this dissertation, we focused on four different problems that are of immense relevance to real-world applications. The goal of this dissertation was to view each of these problems through rigorous theoretical lens.

- Pseudorandom number generation: We presented theoretical modeling that captured practical entropy sources. Further, we presented our constructions which are easily instantiated and based on industry-standard cryptographic hash functions. Our results can be applied to understand the security of the general purpose PRNGs that form a part of operating systems, including Fortuna used in Windows and Yarrow used in MacOS.

- Security of block ciphers: We introduced a new theoretical framework that combines proofs and conjectures to capture security of existing and extensively used block ciphers (AES), providing realistic hardness estimates.

- Public Key Searchable Encryption: We achieved sublinear search time with public key indexing in searchable encryption, and our construction is deployed as a commercially available product.

- Updatable Public Key Encryption: We presented the first constructions that are secure in the standard model, one of which is also the first construction that is believed to be post-quantum secure.

There are several interesting open problems in each of these areas of research and we summarize some of them below.

- We introduced Small-box cryptography as a framework and showed its applications for block ciphers and stream ciphers. Are there other symmetric key constructions for which our framework works? Additionally, are there any guardrails for when our framework can actually be applied and used?
- Our construction of Encapsulated Search Index critically leveraged the document specific indexing to achieve sub-linear search time. Can we have universal indexing, public key cryptography, and sublinear search time?
- We presented our standard model constructions of UPKE which have large parameter sizes. Can we optimize the parameter sizes to make it practically implementable? Are there other assumptions from which we can build UPKE?
- We presented a framework to build UPKE from PKE which are simultaneously leakage-resilient and circular secure, is key homomorphic, and message homomorphic. Is there another framework that we can leverage to achieve better parameter choices?

Bibliography

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 553–572, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany.

[ABC+05] Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 205–222, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Heidelberg, Germany.

[ACD+22a] Erik Aronesty, David Cash, Yevgeniy Dodis, Daniel H. Gallancy, Christopher Higley, Harish Karthikeyan, and Oren Tysor. Encapsulated search index: Public-key, sublinear, distributed, and delegatable. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8-11, 2022, Proceedings, Part II*, volume 13178 of *Lecture Notes in Computer Science*, pages 256–285. Springer, 2022.

[ACD+22b] Erik Aronesty, David Cash, Yevgeniy Dodis, Daniel H. Gallancy, Christopher Higley, Harish Karthikeyan, and Oren Tysor. Encapsulated search index: Public-key, sublinear, distributed, and delegatable. Cryptology ePrint Archive, Report 2022/071, 2022. https://eprint.iacr.org/2022/071.

[ACDT20] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Security analysis and improvements for the IETF MLS standard for group messaging. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 248–277, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany.

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 595–618, Santa Barbara, CA, USA, August 16–20, 2009. Springer, Heidelberg, Germany.

[AFL16] Farzaneh Abed, Christian Forler, and Stefan Lucks. Overview of the candidates in the caesar competition for authenticated encryption. *COMPUTER SCIENCE REVIEW*, 22:13–26, 2016.

[AGV09] Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In Omer Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 474–495. Springer, Heidelberg, Germany, March 15–17, 2009.

[AJL+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas

Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 483–501, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

[AMMR18] Shashank Agrawal, Payman Mohassel, Pratyay Mukherjee, and Peter Rindal. DiSE: Distributed symmetric-key encryption. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 1993–2010, Toronto, ON, Canada, October 15–19, 2018. ACM Press.

[And97] Ross Anderson. Invited lecture. Fourth Annual Conference on Computer and Communications Security, ACM, 1997.

[AP12] Jacob Alperin-Sheriff and Chris Peikert. Circular and KDM security for identity-based encryption. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012: 15th International Conference on Theory and Practice of Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 334–352, Darmstadt, Germany, May 21–23, 2012. Springer, Heidelberg, Germany.

[App13] Benny Applebaum. Cryptographic hardness of random local functions–survey. In Amit Sahai, editor, *Theory of Cryptography*, pages 599–599, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[App14] Benny Applebaum. Key-dependent message security: Generic amplification and completeness. *Journal of Cryptology*, 27(3):429–451, July 2014.

[AR00] Michel Abdalla and Leonid Reyzin. A new forward-secure digital signature scheme. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 116–129, Kyoto, Japan, December 3–7, 2000. Springer, Heidelberg, Germany.

[Ata22] Atakama. Secure your files multi-factor encryption, 2022. https://www.atakama.com/.

[BB04] Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity based encryption without random oracles. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 223–238, Interlaken, Switzerland, May 2–6, 2004. Springer, Heidelberg, Germany.

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 440–456, Aarhus, Denmark, May 22–26, 2005. Springer, Heidelberg, Germany.

[BBH06] Dan Boneh, Xavier Boyen, and Shai Halevi. Chosen ciphertext secure public key threshold encryption without random oracles. In David Pointcheval, editor, *Topics in Cryptology – CT-RSA 2006*, volume 3860 of *Lecture Notes in Computer Science*, pages 226–243, San Jose, CA, USA, February 13–17, 2006. Springer, Heidelberg, Germany.

[BBK14] Alex Biryukov, Charles Bouillaguet, and Dmitry Khovratovich. Cryptographic schemes based on the ASASA structure: Black-box, white-box, and public-key (extended abstract). In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 63–84, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014. Springer, Heidelberg, Germany.

[BC04] Steven M. Bellovin and William R. Cheswick. Privacy-enhanced searches using encrypted bloom filters. Cryptology ePrint Archive, Report 2004/022, 2004. https://eprint.iacr.org/2004/022.

[BDGJ20] Colin Boyd, Gareth T. Davies, Kristian Gjøsteen, and Yao Jiang. Fast and secure updatable encryption. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 464–493, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany.

[BDOP04] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522, Interlaken, Switzerland, May 2–6, 2004. Springer, Heidelberg, Germany.

[BDPV10] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge-based pseudo-random number generators. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems – CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 33–47, Santa Barbara, CA, USA, August 17–20, 2010. Springer, Heidelberg, Germany.

[BEKS20] Dan Boneh, Saba Eskandarian, Sam Kim, and Maurice Shih. Improving speed and security in updatable encryption schemes. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part III*, volume 12493 of *Lecture Notes in Computer Science*, pages 559–589, Daejeon, South Korea, December 7–11, 2020. Springer, Heidelberg, Germany.

[BF03] Dan Boneh and Matthew Franklin. Identity-based encryption from the weil pairing. *SIAM Journal on Computing*, 32(3):586–615, 2003.

[BG10] Zvika Brakerski and Shafi Goldwasser. Circular and leakage resilient public-key encryption under subgroup indistinguishability - (or: Quadratic residuosity strikes

back). In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 1–20, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany.

[BGK11] Zvika Brakerski, Shafi Goldwasser, and Yael Tauman Kalai. Black-box circular-secure encryption beyond affine functions. In Yuval Ishai, editor, *TCC 2011: 8th Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer Science*, pages 201–218, Providence, RI, USA, March 28–30, 2011. Springer, Heidelberg, Germany.

[BH05] Boaz Barak and Shai Halevi. A model and architecture for pseudo-random generation with applications to /dev/random. In Vijayalakshmi Atluri, Catherine Meadows, and Ari Juels, editors, *ACM CCS 2005: 12th Conference on Computer and Communications Security*, pages 203–212, Alexandria, Virginia, USA, November 7–11, 2005. ACM Press.

[BHHI10] Boaz Barak, Iftach Haitner, Dennis Hofheinz, and Yuval Ishai. Bounded key-dependent message security. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 423–444, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany.

[BHHO08] Dan Boneh, Shai Halevi, Michael Hamburg, and Rafail Ostrovsky. Circular-secure encryption from decision Diffie-Hellman. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 108–125, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Heidelberg, Germany.

[BHK13] Mihir Bellare, Viet Tung Hoang, and Sriram Keelveedhi. Instantiating random oracles via UCEs. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 398–415, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[BIW04] Boaz Barak, Russell Impagliazzo, and Avi Wigderson. Extracting randomness using few independent sources. In *45th Annual Symposium on Foundations of Computer Science*, pages 384–393, Rome, Italy, October 17–19, 2004. IEEE Computer Society Press.

[BK] Alex Biryukov and Dmitry Khovratovich. Decomposition attack on SASASASAS. Available at http://eprint.iacr.org/2015/646.

[BK12a] Elaine Barker and John Kelsey. NIST Special Publication 800-90A (A revision of SP 800-90) Recommendation for random number generation using deterministic random bit generators. https://csrc.nist.gov/publications/detail/sp/800-90a/rev-1/final, 2012.

[BK12b] Elaine Barker and John Kelsey. Recommendation for random number generation using deterministic random bit generators. NIST Special Publication 800-90A, 2012.

[BKL+12] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, François-Xavier Standaert, John P. Steinberger, and Elmar Tischhauser. Key-alternating ciphers in a provable setting: Encryption using a small number of public permutations - (extended abstract). In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 45–62, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

[BKOS07] Dan Boneh, Eyal Kushilevitz, Rafail Ostrovsky, and William E. Skeith III. Public key encryption that allows PIR queries. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 50–67, Santa Barbara, CA, USA, August 19–23, 2007. Springer, Heidelberg, Germany.

[BLMR13] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic PRFs and their applications. In Ran Canetti and Juan A. Garay, ed-

itors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 410–428, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[Blo70] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532, Gold Coast, Australia, December 9–13, 2001. Springer, Heidelberg, Germany.

[BLSV18] Zvika Brakerski, Alex Lombardi, Gil Segev, and Vinod Vaikuntanathan. Anonymous IBE, leakage resilience and circular security from new assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 535–564, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

[Blu86] Manuel Blum. Independent unbiased coin flips from a correlated biased source-a finite stae markov chain. *Combinatorica*, 6(2):97–108, 1986.

[BM99] Mihir Bellare and Sara K. Miner. A forward-secure digital signature scheme. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 431–448, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Heidelberg, Germany.

[BMW05] Xavier Boyen, Qixiang Mei, and Brent Waters. Direct chosen ciphertext security from identity-based techniques. In Vijayalakshmi Atluri, Catherine Meadows, and Ari Juels, editors, *ACM CCS 2005: 12th Conference on Computer and Communica-*

tions Security, pages 320–329, Alexandria, Virginia, USA, November 7–11, 2005. ACM Press.

[BRS03] John Black, Phillip Rogaway, and Thomas Shrimpton. Encryption-scheme security in the presence of key-dependent messages. In Kaisa Nyberg and Howard M. Heys, editors, *SAC 2002: 9th Annual International Workshop on Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 62–75, St. John's, Newfoundland, Canada, August 15–16, 2003. Springer, Heidelberg, Germany.

[BS10] Alex Biryukov and Adi Shamir. Structural cryptanalysis of sasas. *J. Cryptology*, 23:505–518, 2010.

[BSNS08] Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. Public key encryption with keyword search revisited. In Osvaldo Gervasi, Beniamino Murgante, Antonio Laganà, David Taniar, Youngsong Mun, and Marina L. Gavrilova, editors, *Computational Science and Its Applications – ICCSA 2008*, pages 1249–1259, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[BST03] Boaz Barak, Ronen Shaltiel, and Eran Tromer. True random number generators secure in a changing environment. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 166–180, Cologne, Germany, September 8–10, 2003. Springer, Heidelberg, Germany.

[BY03] Mihir Bellare and Bennet S. Yee. Forward-security in private-key cryptography. In Marc Joye, editor, *Topics in Cryptology – CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 1–18, San Francisco, CA, USA, April 13–17, 2003. Springer, Heidelberg, Germany.

[CDG18] Sandro Coretti, Yevgeniy Dodis, and Siyao Guo. Non-uniform bounds in the random-permutation, ideal-cipher, and generic-group models. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 693–721, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

[CDK+18] Benoît Cogliati, Yevgeniy Dodis, Jonathan Katz, Jooyoung Lee, John P. Steinberger, Aishwarya Thiruvengadam, and Zhe Zhang. Provable security of (tweakable) block ciphers based on substitution-permutation networks. In *Advances in Cryptology – CRYPTO 2018*, volume 10991 of *Lecture Notes in Computer Science*, pages 722–753. Springer, 2018.

[CDK+22] Sandro Coretti, Yevgeniy Dodis, Harish Karthikeyan, Noah Stephens-Davidowitz, and Stefano Tessaro. On seedless prngs and premature next. In Dana Dachman-Soled, editor, *3rd Conference on Information-Theoretic Cryptography, ITC 2022, July 5-7, 2022*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[CDKT19a] Sandro Coretti, Yevgeniy Dodis, Harish Karthikeyan, and Stefano Tessaro. Seedless Fruit is the sweetest: Random number generation, revisited. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 205–234, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.

[CDKT19b] Sandro Coretti, Yevgeniy Dodis, Harish Karthikeyan, and Stefano Tessaro. Seedless Fruit is the sweetest: Random number generation, revisited. Cryptology ePrint Archive, Report 2019/198, 2019. https://eprint.iacr.org/2019/198.

[CG85] Benny Chor and Oded Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity (extended abstract). In *26th Annual*

Symposium on Foundations of Computer Science, pages 429–442, Portland, Oregon, October 21–23, 1985. IEEE Computer Society Press.

[CG88] Benny Chor and Oded Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM J. Comput.*, 17(2):230–261, 1988.

[CGH+85] Benny Chor, Oded Goldreich, Johan Håstad, Joel Friedman, Steven Rudich, and Roman Smolensky. The bit extraction problem of t-resilient functions (preliminary version). In *26th Annual Symposium on Foundations of Computer Science*, pages 396–407, Portland, Oregon, October 21–23, 1985. IEEE Computer Society Press.

[CGKO06] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006: 13th Conference on Computer and Communications Security*, pages 79–88, Alexandria, Virginia, USA, October 30 – November 3, 2006. ACM Press.

[CGPR15] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015: 22nd Conference on Computer and Communications Security*, pages 668–679, Denver, CO, USA, October 12–16, 2015. ACM Press.

[CHK03] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 255–271, Warsaw, Poland, May 4–8, 2003. Springer, Heidelberg, Germany.

[CHK04] Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*,

pages 207–222, Interlaken, Switzerland, May 2–6, 2004. Springer, Heidelberg, Germany.

[CHKP12] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. *Journal of Cryptology*, 25(4):601–639, October 2012.

[CHS05] Ran Canetti, Shai Halevi, and Michael Steiner. Hardness amplification of weakly verifiable puzzles. In Joe Kilian, editor, *TCC 2005: 2nd Theory of Cryptography Conference*, volume 3378 of *Lecture Notes in Computer Science*, pages 17–33, Cambridge, MA, USA, February 10–12, 2005. Springer, Heidelberg, Germany.

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 93–118, Innsbruck, Austria, May 6–10, 2001. Springer, Heidelberg, Germany.

[CLLY10] Kai-Min Chung, Feng-Hao Liu, Chi-Jen Lu, and Bo-Yin Yang. Efficient string-commitment from weak bit-commitment. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 268–282, Singapore, December 5–9, 2010. Springer, Heidelberg, Germany.

[Clo] Cloudflare. Cloudflare Randomness Beacon docs. https://developers.cloudflare.com/randomness-beacon/.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[CM05] Yan-Cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In John Ioannidis, Angelos Keromytis, and Moti Yung,

editors, *ACNS 05: 3rd International Conference on Applied Cryptography and Network Security*, volume 3531 of *Lecture Notes in Computer Science*, pages 442–455, New York, NY, USA, June 7–10, 2005. Springer, Heidelberg, Germany.

[CNE+14] Stephen Checkoway, Ruben Niederhagen, Adam Everspaugh, Matthew Green, Tanja Lange, Thomas Ristenpart, Daniel J. Bernstein, Jake Maskiewicz, Hovav Shacham, and Matthew Fredrikson. On the practical exploitability of dual EC in TLS implementations. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 319–335, 2014.

[Cor20] Corestar. corestario/tendermint, October 2020. original-date: 2018-12-19T13:33:15Z.

[CRS+07] Ran Canetti, Ronald L. Rivest, Madhu Sudan, Luca Trevisan, Salil P. Vadhan, and Hoeteck Wee. Amplifying collision resistance: A complexity-theoretic treatment. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 264–283, Santa Barbara, CA, USA, August 19–23, 2007. Springer, Heidelberg, Germany.

[CS06] Debrup Chakraborty and Palash Sarkar. A new mode of encryption providing a tweakable strong pseudo-random permutation. In Matthew J. B. Robshaw, editor, *Fast Software Encryption – FSE 2006*, volume 4047 of *Lecture Notes in Computer Science*, pages 293–309, Graz, Austria, March 15–17, 2006. Springer, Heidelberg, Germany.

[CS14] Shan Chen and John P. Steinberger. Tight security bounds for key-alternating ciphers. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 327–350, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.

[CZ16] Eshan Chattopadhyay and David Zuckerman. Explicit two-source extractors and resilient functions. In Daniel Wichs and Yishay Mansour, editors, *48th Annual ACM*

Symposium on Theory of Computing, pages 670–683, Cambridge, MA, USA, June 18–21, 2016. ACM Press.

[DAO19] DAOBet. DAOBet (ex — DAO.Casino) to Deliver On-Chain Random Beacon Based on BLS Cryptography, May 2019. https://daobet.org/blog/on-chain-random-generator/.

[DBS04] Ratna Dutta, Rana Barua, and Palash Sarkar. Pairing-based cryptographic protocols : a survey. Cryptology ePrint Archive, Report 2004/064, 2004. http://eprint.iacr.org/2004/064/.

[DDKL15] Itai Dinur, Orr Dunkelman, Thorsten Kranz, and Gregor Leander. Decomposing the ASASA block cipher construction. Cryptology ePrint Archive, Report 2015/507, 2015. https://eprint.iacr.org/2015/507.

[DG17a] Nico Döttling and Sanjam Garg. From selective IBE to full IBE and selective HIBE. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 372–408, Baltimore, MD, USA, November 12–15, 2017. Springer, Heidelberg, Germany.

[DG17b] Nico Döttling and Sanjam Garg. Identity-based encryption from the Diffie-Hellman assumption. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 537–569, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[DGH+04] Yevgeniy Dodis, Rosario Gennaro, Johan Håstad, Hugo Krawczyk, and Tal Rabin. Randomness extraction and key derivation using the CBC, cascade and HMAC modes. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 494–510, Santa Barbara, CA, USA, August 15–19, 2004. Springer, Heidelberg, Germany.

[DGK+10] Yevgeniy Dodis, Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Public-key encryption schemes with auxiliary inputs. In Daniele Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 361–381, Zurich, Switzerland, February 9–11, 2010. Springer, Heidelberg, Germany.

[DGSX21a] Yevgeniy Dodis, Siyao Guo, Noah Stephens-Davidowitz, and Zhiye Xie. No time to hash: On super-efficient entropy accumulation. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part IV*, volume 12828 of *Lecture Notes in Computer Science*, pages 548–576, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany.

[DGSX21b] Yevgeniy Dodis, Siyao Guo, Noah Stephens-Davidowitz, and Zhiye Xie. Online linear extractors for independent sources. In Stefano Tessaro, editor, *2nd Conference on Information-Theoretic Cryptography, ITC 2021, July 23-26, 2021, Virtual Conference*, volume 199 of *LIPIcs*, pages 14:1–14:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[DIJK09] Yevgeniy Dodis, Russell Impagliazzo, Ragesh Jaiswal, and Valentine Kabanets. Security amplification for interactive cryptographic primitives. In Omer Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 128–145. Springer, Heidelberg, Germany, March 15–17, 2009.

[Div14] NIST Computer Security Division. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. FIPS Publication 202, National Institute of Standards and Technology, U.S. Department of Commerce, May 2014.

[DJMW12] Yevgeniy Dodis, Abhishek Jain, Tal Moran, and Daniel Wichs. Counterexamples to hardness amplification beyond negligible. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 476–493, Taormina, Sicily, Italy, March 19–21, 2012. Springer, Heidelberg, Germany.

[DKS12] Orr Dunkelman, Nathan Keller, and Adi Shamir. Minimalism in cryptography: The Even-Mansour scheme revisited. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 336–354, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

[DKW21] Yevgeniy Dodis, Harish Karthikeyan, and Daniel Wichs. Updatable public key encryption in the standard model. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography - 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8-11, 2021, Proceedings, Part III*, volume 13044 of *Lecture Notes in Computer Science*, pages 254–285. Springer, 2021.

[DKW22a] Yevgeniy Dodis, Harish Karthikeyan, and Daniel Wichs. Small-Box Cryptography. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, volume 215 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 56:1–56:25, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[DKW22b] Yevgeniy Dodis, Harish Karthikeyan, and Daniel Wichs. Updatable public key encryption in the standard model. *IACR Cryptol. ePrint Arch.*, page 68, 2022.

[DNR04] Cynthia Dwork, Moni Naor, and Omer Reingold. Immunizing encryption schemes from decryption errors. In Christian Cachin and Jan Camenisch, editors, *Advances*

in Cryptology – EUROCRYPT 2004, volume 3027 of *Lecture Notes in Computer Science*, pages 342–360, Interlaken, Switzerland, May 2–6, 2004. Springer, Heidelberg, Germany.

[Dod03] Yevgeniy Dodis. Efficient construction of (distributed) verifiable random functions. In Yvo Desmedt, editor, *PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 1–17, Miami, FL, USA, January 6–8, 2003. Springer, Heidelberg, Germany.

[DPR+13] Yevgeniy Dodis, David Pointcheval, Sylvain Ruhault, Damien Vergnaud, and Daniel Wichs. Security analysis of pseudo-random number generators with input: /dev/random is not robust. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013: 20th Conference on Computer and Communications Security*, pages 647–658, Berlin, Germany, November 4–8, 2013. ACM Press.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[DRV12] Yevgeniy Dodis, Thomas Ristenpart, and Salil P. Vadhan. Randomness condensers for efficiently samplable, seed-dependent sources. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 618–635, Taormina, Sicily, Italy, March 19–21, 2012. Springer, Heidelberg, Germany.

[DSSL16] Yevgeniy Dodis, Martijn Stam, John P. Steinberger, and Tianren Liu. Indifferentiability of confusion-diffusion networks. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 679–704, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.

[DSSW14] Yevgeniy Dodis, Adi Shamir, Noah Stephens-Davidowitz, and Daniel Wichs. How to eat your entropy and have it too - optimal recovery strategies for compromised RNGs. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 37–54, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.

[DSSW17] Yevgeniy Dodis, Adi Shamir, Noah Stephens-Davidowitz, and Daniel Wichs. How to eat your entropy and have it too: Optimal recovery strategies for compromised rngs. *Algorithmica*, 79(4):1196–1232, 2017.

[DTT10] Anindya De, Luca Trevisan, and Madhur Tulsiani. Time space tradeoffs for attacks against one-way functions and PRGs. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 649–665, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany.

[DWP00] Dawn Xiaoding Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Proceeding 2000 IEEE Symposium on Security and Privacy. S P 2000*, pages 44–55, 2000.

[DY05] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In Serge Vaudenay, editor, *PKC 2005: 8th International Workshop on Theory and Practice in Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 416–431, Les Diablerets, Switzerland, January 23–26, 2005. Springer, Heidelberg, Germany.

[EM93] Shimon Even and Yishay Mansour. A construction of a cipher from a single pseudorandom permutation. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *Advances in Cryptology – ASIACRYPT'91*, volume 739 of *Lecture Notes in Com-*

puter Science, pages 210–224, Fujiyoshida, Japan, November 11–14, 1993. Springer, Heidelberg, Germany.

[EPRS17] Adam Everspaugh, Kenneth G. Paterson, Thomas Ristenpart, and Samuel Scott. Key rotation for authenticated encryption. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 98–129, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[ESC05] D. Eastlake, J. Schiller, and S. Crocker. *RFC 4086 - Randomness Requirements for Security*, June 2005.

[Fei73] Horst Feistel. Cryptography and computer privacy. *Scientific American*, 228(5):15–23, 1973.

[Fel88] Frank A. Feldman. Fast spectral tests for measuring nonrandomness and the DES. In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 243–254, Santa Barbara, CA, USA, August 16–20, 1988. Springer, Heidelberg, Germany.

[Fer13] Niels Ferguson. Private communication, 2013.

[Fer19] Niels Ferguson. The windows 10 random number generation infrastructure, Oct 2019.

[FS03] Niels Ferguson and Bruce Schneier. *Practical cryptography*. Wiley, 2003.

[Gal01] Steven D. Galbraith. Supersingular curves in cryptography. *Lecture Notes in Computer Science*, 2248:495–513, 2001.

[GJKR07] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, January 2007.

[GLOW20] David Galindo, Jia Liu, Mihai Ordean, and Jin-Mann Wong. Fully distributed verifiable random functions and their application to decentralised random beacons. Cryptology ePrint Archive, Report 2020/096, 2020. https://eprint.iacr.org/2020/096.

[GNP+15] Sharon Goldberg, Moni Naor, Dimitrios Papadopoulos, Leonid Reyzin, Sachin Vasant, and Asaf Ziv. NSEC5: Provably preventing DNSSEC zone enumeration. In *ISOC Network and Distributed System Security Symposium – NDSS 2015*, San Diego, CA, USA, February 8–11, 2015. The Internet Society.

[Goh03] Eu-Jin Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. https://eprint.iacr.org/2003/216.

[Gol95] Oded Goldreich. Foundations of cryptography (fragments of a book). Electronic Colloquium on Computational Complexity, 1995.

[Gol11] Oded Goldreich. *Candidate One-Way Functions Based on Expander Graphs*, page 76–87. Springer-Verlag, Berlin, Heidelberg, 2011.

[GPR14] Peter Gaži, Krzysztof Pietrzak, and Michal Rybár. The exact PRF-security of NMAC and HMAC. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 113–130, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, edi-

tors, *40th Annual ACM Symposium on Theory of Computing*, pages 197–206, Victoria, BC, Canada, May 17–20, 2008. ACM Press.

[GRPV20] Sharon Goldberg, Leonid Reyzin, Dimitrios Papadopoulos, and Jan Včelák. Verifiable Random Functions (VRFs). Internet-Draft draft-irtf-cfrg-vrf-07, Internet Engineering Task Force, June 2020. Work in Progress.

[GS02] Craig Gentry and Alice Silverberg. Hierarchical ID-based cryptography. In Yuliang Zheng, editor, *Advances in Cryptology – ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 548–566, Queenstown, New Zealand, December 1–5, 2002. Springer, Heidelberg, Germany.

[GT16] Peter Gazi and Stefano Tessaro. Provably robust sponge-based PRNGs and KDFs. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 87–116, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.

[Hal07] Shai Halevi. Invertible universal hashing and the TET encryption mode. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 412–429, Santa Barbara, CA, USA, August 19–23, 2007. Springer, Heidelberg, Germany.

[HL02] Jeremy Horwitz and Ben Lynn. Toward hierarchical identity-based encryption. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 466–481, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Heidelberg, Germany.

[HLW06] Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *BULL. AMER. MATH. SOC.*, 43(4):439–561, 2006.

[HN06] Danny Harnik and Moni Naor. On everlasting security in the hybrid bounded storage model. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP 2006: 33rd International Colloquium on Automata, Languages and Programming, Part II*, volume 4052 of *Lecture Notes in Computer Science*, pages 192–203, Venice, Italy, July 10–14, 2006. Springer, Heidelberg, Germany.

[HT16] Viet Tung Hoang and Stefano Tessaro. Key-alternating ciphers and key-length extension: Exact bounds and multi-user security. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 3–32, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

[Hut16a] Daniel Hutchinson. A robust and sponge-like PRNG with improved efficiency. In Roberto Avanzi and Howard M. Heys, editors, *SAC 2016: 23rd Annual International Workshop on Selected Areas in Cryptography*, volume 10532 of *Lecture Notes in Computer Science*, pages 381–398, St. John's, NL, Canada, August 10–12, 2016. Springer, Heidelberg, Germany.

[Hut16b] Daniel Hutchinson. A robust and sponge-like PRNG with improved efficiency. Cryptology ePrint Archive, Report 2016/886, 2016. https://eprint.iacr.org/2016/886.

[IK01] Tetsu Iwata and Kaoru Kurosawa. On the pseudorandomness of the AES finalists - RC6 and Serpent. In Bruce Schneier, editor, *Fast Software Encryption – FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 231–243, New York, NY, USA, April 10–12, 2001. Springer, Heidelberg, Germany.

[ILL89] Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Pseudo-random generation from one-way functions (extended abstracts). In *21st Annual ACM Symposium on Theory of Computing*, pages 12–24, Seattle, WA, USA, May 15–17, 1989. ACM Press.

[IR01] Gene Itkis and Leonid Reyzin. Forward-secure signatures with optimal signing and verifying. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 332–354, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

[ISO11] Information technology - Security techniques - Random bit generation. ISO/IEC18031:2011, 2011.

[JKPT12] Abhishek Jain, Stephan Krenn, Krzysztof Pietrzak, and Aris Tentes. Commitments and efficient zero-knowledge proofs from learning parity with noise. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 663–680, Beijing, China, December 2–6, 2012. Springer, Heidelberg, Germany.

[JMM19] Daniel Jost, Ueli Maurer, and Marta Mularczyk. Efficient ratcheting: Almost-optimal guarantees for secure messaging. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 159–188, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany.

[JN01] Antoine Joux and Kim Nguyen. Separating Decision Diffie-Hellman from Diffie-Hellman in cryptographic groups. Cryptology ePrint Archive, Report 2001/003, 2001. `http://eprint.iacr.org/2001/003/`.

[JS18] Joseph Jaeger and Igors Stepanovs. Optimal channel security against fine-grained state compromise: The safety of messaging. In Hovav Shacham and Alexandra

Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 33–62, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

[Kee20] Keep. The Keep Random Beacon: An Implementation of a Threshold Relay, 2020. https://docs.keep.network/random-beacon/.

[Kil11] Killmann, W. and Schindler, W. A proposal for: Functionality classes for random number generators. AIS 20 / AIS31, 2011.

[KLR19] Michael Klooß, Anja Lehmann, and Andy Rupp. (R)CCA secure updatable encryption with integrity protection. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 68–99, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany.

[KM13] Veronika Kuchta and Mark Manulis. Unique aggregate signatures with applications to distributed verifiable random functions. In Michel Abdalla, Cristina Nita-Rotaru, and Ricardo Dahab, editors, *CANS 13: 12th International Conference on Cryptology and Network Security*, volume 8257 of *Lecture Notes in Computer Science*, pages 251–270, Paraty, Brazil, November 20–22, 2013. Springer, Heidelberg, Germany.

[KPR12] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012: 19th Conference on Computer and Communications Security*, pages 965–976, Raleigh, NC, USA, October 16–18, 2012. ACM Press.

[KR01] Joe Kilian and Phillip Rogaway. How to protect DES against exhaustive key search (an analysis of DESX). *Journal of Cryptology*, 14(1):17–35, January 2001.

[KR03] Anton Kozlov and Leonid Reyzin. Forward-secure signatures with fast key update. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN 02: 3rd International Conference on Security in Communication Networks*, volume 2576 of *Lecture Notes in Computer Science*, pages 241–256, Amalfi, Italy, September 12–13, 2003. Springer, Heidelberg, Germany.

[KR19] Yael Tauman Kalai and Leonid Reyzin. A survey of leakage-resilient cryptography. Cryptology ePrint Archive, Report 2019/302, 2019. https://eprint.iacr.org/2019/302.

[Kra10] Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 631–648, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany.

[KRVZ11] Jesse Kamp, Anup Rao, Salil P. Vadhan, and David Zuckerman. Deterministic extractors for small-space sources. *J. Comput. Syst. Sci.*, 77(1):191–220, 2011.

[KSD13] Cameron F. Kerry, Acting Secretary, and Charles Romine Director. Fips pub 186-4 federal information processing standards publication digital signature standard (dss), 2013.

[KSF99] John Kelsey, Bruce Schneier, and Niels Ferguson. Yarrow-160: Notes on the design and analysis of the yarrow cryptographic pseudorandom number generator. In *In Sixth Annual Workshop on Selected Areas in Cryptography*, pages 13–33. Springer, 1999.

[KSWH98] John Kelsey, Bruce Schneier, David Wagner, and Chris Hall. Cryptanalytic attacks on pseudorandom number generators. In Serge Vaudenay, editor, *Fast Software En-*

cryption – FSE'98, volume 1372 of *Lecture Notes in Computer Science*, pages 168–188, Paris, France, March 23–25, 1998. Springer, Heidelberg, Germany.

[LLS89] David Lichtenstein, Nathan Linial, and Michael E. Saks. Some extremal problems arising form discrete control processes. *Combinatorica*, 9(3):269–287, 1989.

[LR88] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2), 1988.

[LT18] Anja Lehmann and Björn Tackmann. Updatable encryption with post-compromise security. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 685–716, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

[Lys02] Anna Lysyanskaya. Unique signatures and verifiable random functions from the DH-DDH separation. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 597–612, Santa Barbara, CA, USA, August 18–22, 2002. Springer, Heidelberg, Germany.

[M14] John M. Intel digital random number generator (DRNG) software implementation guide. https://software.intel.com/en-us/articles/intel-digital-random-number-generator-drng-software-implementation-guide, 2014.

[MMM02] Tal Malkin, Daniele Micciancio, and Sara K. Miner. Efficient generic forward-secure signatures with an unbounded number of time periods. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 400–417, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Heidelberg, Germany.

[MPR07] Ueli M. Maurer, Krzysztof Pietrzak, and Renato Renner. Indistinguishability amplification. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 130–149, Santa Barbara, CA, USA, August 19–23, 2007. Springer, Heidelberg, Germany.

[MRH04] Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39, Cambridge, MA, USA, February 19–21, 2004. Springer, Heidelberg, Germany.

[MRV99] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science*, pages 120–130, New York, NY, USA, October 17–19, 1999. IEEE Computer Society Press.

[MTY11] Tal Malkin, Isamu Teranishi, and Moti Yung. Efficient circuit-size independent public key encryption with KDM security. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 507–526, Tallinn, Estonia, May 15–19, 2011. Springer, Heidelberg, Germany.

[MV15] Eric Miles and Emanuele Viola. Substitution-permutation networks, pseudorandom functions, and natural proofs. *J. ACM*, 62(6):46:1–46:29, 2015.

[NPR99] Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and KDCs. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 327–346, Prague, Czech Republic, May 2–6, 1999. Springer, Heidelberg, Germany.

[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th Annual Symposium on Foundations of Computer Science*,

pages 458–467, Miami Beach, Florida, October 19–22, 1997. IEEE Computer Society Press.

[NR99] Moni Naor and Omer Reingold. On the construction of pseudorandom permutations: Luby-Rackoff revisited. *Journal of Cryptology*, 12(1):29–66, January 1999.

[NS09] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 18–35, Santa Barbara, CA, USA, August 16–20, 2009. Springer, Heidelberg, Germany.

[NY15] Moni Naor and Eylon Yogev. Tight bounds for sliding bloom filters. *Algorithmica*, 73(4):652–672, December 2015.

[NZ96] Noam Nisan and David Zuckerman. Randomness is linear in space. *J. Comput. Syst. Sci.*, 52(1):43–52, 1996.

[Pat09] Jacques Patarin. The "coefficients H" technique (invited talk). In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *SAC 2008: 15th Annual International Workshop on Selected Areas in Cryptography*, volume 5381 of *Lecture Notes in Computer Science*, pages 328–345, Sackville, New Brunswick, Canada, August 14–15, 2009. Springer, Heidelberg, Germany.

[PPR05] Anna Pagh, Rasmus Pagh, and S. Srinivasa Rao. An optimal bloom filter replacement. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, page 823–829, USA, 2005. Society for Industrial and Applied Mathematics.

[PR04] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122 – 144, 2004.

[PR18] Bertram Poettering and Paul Rösler. Towards bidirectional ratcheted key exchange. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 3–32, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

[RB08] Matthew Robshaw and Olivier Billet, editors. *New Stream Cipher Designs: The ESTREAM Finalists*. Springer-Verlag, Berlin, Heidelberg, 2008.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 84–93, Baltimore, MA, USA, May 22–24, 2005. ACM Press.

[RPSL09] Hyun Sook Rhee, Jong Hwan Park, Willy Susilo, and Dong Hoon Lee. Improved searchable public key encryption with designated tester. In Wanqing Li, Willy Susilo, Udaya Kiran Tupakula, Reihaneh Safavi-Naini, and Vijay Varadharajan, editors, *ASIACCS 09: 4th ACM Symposium on Information, Computer and Communications Security*, pages 376–379, Sydney, Australia, March 10–12, 2009. ACM Press.

[Sch80] Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27:701–717, 1980.

[SF07] Dan Shumow and Niels Ferguson. On the possibility of a back door in the nist sp800-90 dual ec prng. *CRYPTO Rump Session*, 2007.

[Sha49] Claude E. Shannon. Communication theory of secrecy systems. *Bell Systems Technical Journal*, 28(4):656–715, 1949.

[Sha79] A. Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, 1979.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. *Lecture Notes in Computer Science*, 1233:256–266, 1997.

[Sho01] Victor Shoup. A proposal for an ISO standard for public key encryption. Cryptology ePrint Archive, Report 2001/112, 2001. https://eprint.iacr.org/2001/112.

[SJSW19] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar Weippl. ETHDKG: Distributed key generation with Ethereum smart contracts. Cryptology ePrint Archive, Report 2019/985, 2019. https://eprint.iacr.org/2019/985.

[SR12] John Bryson Secretary and Charles H. Romine. Fips pub 180-4 federal information processing standards publication secure hash standard (shs), 2012.

[SS06] Berry Schoenmakers and Andrey Sidorenko. Cryptanalysis of the dual elliptic curve pseudorandom generator. Cryptology ePrint Archive, Report 2006/190, 2006. https://eprint.iacr.org/2006/190.

[ST15] Thomas Shrimpton and R. Seth Terashima. A provable-security analysis of Intel's secure key RNG. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 77–100, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

[ST17] Pratik Soni and Stefano Tessaro. Public-seed pseudorandom permutations. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 412–441, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany.

[Tes11] Stefano Tessaro. Security amplification for the cascade of arbitrarily weak PRPs: Tight bounds via the interactive hardcore lemma. In Yuval Ishai, editor, *TCC 2011: 8th Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer*

Science, pages 37–54, Providence, RI, USA, March 28–30, 2011. Springer, Heidelberg, Germany.

[TV00] Luca Trevisan and Salil P. Vadhan. Extracting randomness from samplable distributions. In *41st Annual Symposium on Foundations of Computer Science*, pages 32–42, Redondo Beach, CA, USA, November 12–14, 2000. IEEE Computer Society Press.

[von51] John von Neumann. Various techniques used in connection with random digits. In A.S. Householder, G.E. Forsythe, and H.H. Germond, editors, *Monte Carlo Method*, pages 36–38. National Bureau of Standards Applied Mathematics Series, 12, Washington, D.C.: U.S. Government Printing Office, 1951.

[Wik21] Wikipedia contributors. /dev/random — Wikipedia, the free encyclopedia, 2021. [Online; accessed 9-January-2022].

[WS19] Joanne Woodage and Dan Shumow. An analysis of NIST SP 800-90A. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 151–180, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany.

[Yao82] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press.

[ZLT+20] Yunhong Zhou, Na Li, Yanmei Tian, Dezhi An, and Licheng Wang. Public key encryption with keyword search in cloud: A survey. *Entropy*, 22(4), 2020.