IACR Communications in Cryptology
ISSN 3006-5496, Vol. 1, No. 4, 46 pages.

https://doi.org/10.62056/a3w7tr-10k

Check for updates

HELP: Everlasting Privacy through Server-Aided Randomness

Yevgeniy Dodis[1], Jiaxin Guan[1], Peter Hall[1] and Alison Lin[2]

[1] New York University, New York, USA
[2] Independent Contributor, USA

Abstract. Everlasting (EL) privacy offers an attractive solution to the Store-Now-Decrypt-Later (SNDL) problem, where future increases in the attacker's capability could break systems which are believed to be secure today. Instead of requiring full information-theoretic security, everlasting privacy allows computationally-secure transmissions of ephemeral secrets, which are only "effective" for a limited periods of time, after which their compromise is provably useless for the SNDL attacker.

In this work we revisit such everlasting privacy model of Dodis and Yeo [DY21] (ITC'21), which we call *Hypervisor EverLasting Privacy* (HELP). HELP is a novel architecture for generating shared randomness using a network of semi-trusted servers (or "hypervisors"), trading the need to store/distribute large shared secrets with the assumptions that it is hard to: (a) simultaneously compromise too many publicly accessible ad-hoc servers; and (b) break a computationally-secure encryption scheme very quickly. While Dodis and Yeo [DY21] presented good HELP solutions in the asymptotic sense, their solutions were concretely expensive and used heavy tools (like large finite fields or gigantic Toeplitz matrices).

We abstract and generalize the HELP architecture to allow for more efficient instantiations, and construct several concretely efficient HELP solutions. Our solutions use elementary cryptographic operations, such as hashing and message authentication. We also prove a very strong composition theorem showing that our EL architecture can use *any* message transmission method which is *computationally-secure* in the Universal Composability (UC) framework. This is the first positive composition result for everlasting privacy, which was otherwise known to suffer from many "non-composition" results (Müller-Quade and Unruh [MU10]; J of Cryptology).

1 Introduction

Public-key cryptography has come a long way from theory to practice, allowing people to securely communicate without expensive key distribution. Unfortunately, public-key cryptography comes at the cost of requiring unproven computational assumptions, which may be falsified some day or broken by more powerful computing in the future. This is especially important in light of the famous *Store-Now-Decrypt-Later* (SNDL) attack [BMV06, MVZJ18], where a computationally bounded attacker can passively store sensitive ciphertexts that it cannot decrypt at the moment. Later, when the power of the attacker increases (through novel cryptanalysis, or say, running Shor's algorithm [Sho94] or Grover's algorithm [Gro96] on a quantum computer), the attacker can then decrypt the stored messages. Indeed, traditional public-key encryption schemes are ill-equipped to combat the SNDL attack, and a lot of attention is given to finding alternative solutions. Post-quantum cryptography (PQC) is a step in this direction, in that it directly addresses

E-mail: dodis@cs.nyu.edu (Yevgeniy Dodis), jiaxin@guan.io (Jiaxin Guan), pf2184@nyu.edu (Peter Hall), colorfly@gmail.com (Alison Lin)

Received: 2024-07-09 Accepted: 2024-12-03

the threat of quantum computers. Unfortunately, PQC is still in its infancy, and the community does not have very high confidence in traditional post-quantum assumptions. For example, the SIKE [FJP14, JF11] public-key encryption was originally selected as one of the finalists to the NIST PQC competition [CCH$^+$22] only to be broken almost immediately after [CD23, MMP$^+$23]. More recently, Chen announced [Che24] a groundbreaking attack on the famous Learning-with-Errors problem [Reg10]. If correct, the attack would likely have devastating consequences to the existing PQC candidates. Fortunately, the community have found a mistake in the proposal (see [Che24]). This mistake took weeks to find, though, and many experts were clearly concerned by the potential break.

While PQC solutions form an important line of defense against SNDL attacks, there is also a need for solutions which offer stronger security and/or confidence to the system's users, even against unforeseen (and even believed to be unlikely) future cryptanalysis. From this perspective, Information-Theoretic (IT) security would be much preferable, as these do not rely on any computational assumption. As a result, IT-security resists advances in computational power, novel cryptanalysis, and the threat of future quantum computers.

Unfortunately, the famous impossibility result of Shannon [Sha49, Dod12] states that IT-security comes at a price: The secret must be at least as large as the message. To encrypt large amounts of data, then, it appears that users must pre-share large amounts of randomness, which is often very onerous and impractical.

Everlasting Privacy. Several communication models have emerged over the years which overcome this limitation, by settling on a novel type of security called *everlasting privacy* [ADR02, DR02, HN06], which lies between the traditional computational and the full IT privacy. First, these models require some type of "outsourcing" of large amounts of shared randomness to one or more semi-trusted entities. We call these *hypervisors*, or just *servers*, in this work. The hypervisors will help communicating parties agree on effectively arbitrary amount of shared randomness — which can be used as a one-time pad, for example, — without the parties needing to pre-share this randomness. We will assume the minimal level of trust from the servers. Concretely, if malicious servers are modeled as active attackers, we need an honest majority (as otherwise users cannot tell which half of the servers is "for real"). And for passive/failing servers, we assume only 1 honest server (clearly optimal). In particular, if *all* the hypervisors are distrusted, the IT models for key distribution will not offer additional benefits beyond computational security (offered by public-key cryptography).

Second, these models assume some realistic, *temporary* limitations on the attacker during the phase that the hypervisors have not erased all the randomness which is no longer needed. Once this temporary phase is over, the attacker can be *completely unbounded*. However, at this point, this powerful adversary will be unable to break the privacy of the system, because the randomness required for this task has already been erased. To state this directly, we can separate into two customizable timelines: (1) "short/medium", where the generated one-time pads need to be reproducible by good parties; (2) "long", by which the pads should not be reconstructible by attackers, provided servers properly erased randomness. In practice, it is not hard to set these thresholds for concrete applications; e.g. (1) could be hours/days, while (2) likely years/decades.

Everlasting privacy serves as a perfect compromise between computational and IT privacy in combating SNDL-style attacks. During a reasonably short window, which is often under the control of the application,[1] one could make realistic assumptions about the attacker's limitations. By bounding how long they must hold, these assumptions are firmly based in reality and can take into account the current state of quantum computers, classical computing capabilities, geographic considerations (e.g., when the servers are well

[1] This could be as short as minutes or even seconds; certainly, less than months or years.

separated), or existing cryptanalysis. And once this short window is gone, no (foreseen or unforeseen) development in any of these dimensions will harm the privacy of the system.

Real-World Considerations. The introduction of the hypervisors overcomes the limitations of public-key cryptography, without the need to pre-share huge keys. However, it is a non-trivial new component, so extra care needs to be taken to ensure that the particular everlasting privacy model is interesting. For starters, Shannon's impossibility result [Sha49] implies that preventing SNDL attacks (and hence achieving everlasting privacy) is *impossible* without pre-shared keys, if the attacker can always observe the communication channels between the user and *all* of the hypervisors. Thus, in all meaningful everlasting privacy models it is essential that *some of these communication channels are assumed (perfectly) secure.* While this assumption is (necessarily!) strong, it seems reasonable in many real-world situations. In essence, performing SDNL attacks against users and several hypervisors is often *exponentially harder* than simply dumping encrypted traffic onto hard disk on a single Alice-to-Bob channel, as we explain below.

First, the attacker needs to actually monitor (or "break into") multiple channels instead of a single channel. Second, if servers can come and go in an ad hoc fashion (which our model will easily allow), it might be hard for the attacker to never miss the introduction of a new server. Third, instead of dumping traffic only corresponding to the single channel of interest, now the attacks need to store the traffic from the sender to all the servers, making it more expensive. Fourth, in our model the users can try to disguise, delay, or spread their access to the servers over time, making it extremely hard for the attacker to even figure out which "ciphertext chunks" are relevant for a given target conversation/ciphertext. And, finally, even if everything fails, the users can always use conventional public-key crypto *on top* of our model, meaning that we did not make things worse for the user, but only harder for the SNDL attacks.

Of course, while we want to make the life of the attacker harder, a given model of hypervisor-assisted everlasting privacy should be convenient/cheap to use. Indeed, to prevent trivial (but very impractical) solutions, hypervisors have to be extremely simple, and *cannot keep growing amount of state which depends on the number of generated keys.* For example, in our models we will not allow trivial solution where a user Alice will actively "route" the message to Bob through a trusted hypervisor — or secret shared with multiple hypervisors.[2] Indeed, this would require each hypervisor to keep the message/share until Bob comes on-line, making its state balloon over time. Additionally, it would require a possibly expensive authentication mechanism between the user and the hypervisors. Instead, we want each server to be either stateless, or "minimally stateful" (see below), and not introduce any expensive setup routines between the user and the servers, so that new servers can be added/removed at will.

Motivated by the above considerations, in this work we primarily focus on a particular model of everlasting privacy from Dodis and Yeo [DY21], which is the only model we know to satisfy the requirements mentioned above, and is arguably the most convenient model for practical deployment. (We survey several everlasting privacy proposals in Section 1.3.) As the authors did not give a catchy name to their model, we term it *Hypervisor EverLasting Privacy* (HELP).

1.1 HELP Model for Everlasting Privacy

HELP is a novel architecture for generating shared randomness using a network of semi-trusted servers (or "hypervisors"), trading the need to store/distribute large shared secrets with the assumptions that it is hard to: (a) simultaneously compromise too many publicly

[2] For example, this rules out solutions based on *Secure Message Transmission* (SMT) techniques [DDWY93].

accessible ad-hoc servers; and (b) break a computationally-secure message transmission scheme very quickly. Below we describe the single-server variant first. As we will see, definition and constructions from the one server setting usually generalize quite easily to the multi-server scenario, while providing more realistic trust assumptions about the servers.

Single-Server HELP [DY21]. In this setting, a single server $\mathcal{S}$ is assumed to be trusted. Moreover, somewhat unrealistically it is assumed that $\mathcal{S}$ has *private channel* with all the users of the system (an assumption which will be significantly relaxed in the multi-server setting). The server does not have any explicit authentication mechanism for the users, and arbitrary users (including the adversary!) can communicate with $\mathcal{S}$.

In the original model of [DY21], the server was truly *stateless*: its state consists of a single long-and-random string X, and nothing beside X is remembered between the calls. Imagine now that the user Alice wants to transmit a message m to user Bob. Alice would choose a random seed S, and send it to $\mathcal{S}$. $\mathcal{S}$ will apply a particular primitive called *doubly-affine extractor* to its string X and the seed S, and obtain a random one-time pad R, which it returned to Alice. Moreover, since R was really extracted from X, it is statistically independent from the seed S. Alice will then use any computationally secure mechanism — e.g., public-key encryption, — to send the tuple $(S, m \oplus R)$ to Bob. Bob would go to the server with the same string S to recover the one-time pad R, and finally get the message.

In the meanwhile, the attacker Eve can come to the server with her own seeds $S_1, \ldots, S_q$, and get its own pads $R_1, \ldots, R_q$. However, since Eve is assumed to be unable to decrypt S from the computationally-secure channel, doubly-affine extractors have a property that all these pads $R_1, \ldots, R_q$ are statistically independent from R (as long as q is not too large). Moreover, after some reasonable amount of time — enough for Bob to contact the server after Alice's initial request — the server is assumed to erase its randomness X forever. Once that happens, even if Eve were to become computationally unbounded, it would be too late to break the privacy of m. All Eve could do is maybe get the correct seed S and the one-time pad $m \oplus R$. But the string X is long gone, and R is independent from the seed S, as well as previously obtained pads $R_1, \ldots, R_q$. Hence, we get everlasting privacy.

Some Limitations of the Model. While the work of [DY21] achieved great asymptotic parameters, their model and solution suffered from a number of drawbacks:

(a) *Trade-off between Access Efficiency and Entropy Waste.* The model had an inherent trade-off between the number of bits (called probe complexity) read from the randomizer X, and the total length of one-time pads requested by all users. For example, if $|m| = \ell$ and $|X| = N$, in order to read at most ℓ/β bits from X to derive R, one had to "waste" at least βN bits from X; namely, the sum of all the one-time pads R returned by the server could not be more than $(1 - \beta)N$. Moreover, the ℓ/β bits read are randomly dispersed over X.[3]

(b) *Computational Efficiency.* The doubly-affine extractors from Dodis and Yeo [DY21], while elegant and "polynomial-time", required pretty heavy computation, and non-standard libraries. Concretely, to extract a 1Mb secret (say, to encrypt a low-resolution picture), one would either need a finite field multiplication of size several megabytes (depending on β and other efficiency parameters), or a Toeplitz matrix-vector multiplication of similar size. Both of which can hardly be done efficiently by the server, especially at scale.

(c) *Restricted Everlasting Privacy.* [DY21] first defined an information-theoretically (IT) secure version of doubly-affine extractors, which effectively corresponds to an ideal

[3] And the concrete number were worse; for example, to derive a 512-bit pad, while sacrificing half of the 10Gb randomizer X (so $\beta = 0.5$, and only 5Gb of randomness can be derived overall), the scheme of [DY21] has to access 1527 individual bits of X, which is $\sim 50\%$ worse than reading $512/0.5 = 1024$ bits.

channel between Alice and Bob to transmit $(S, m \oplus R)$. Of course, this is not directly interesting for everlasting privacy, as then Alice could have simply send m over this ideal channel. Nevertheless, [DY21] showed a limited type of *composition theorem*, which implies a restrictive type of everlasting privacy. Concretely, instead of being transmitted over a computationally secure channel, together with $m \oplus R$, the seed S for their IT-solution could be generated by a computationally secure key agreement protocol run *before the* HELP *instance is even initialized.* Moreover, no additional computational leakage about S could happen afterwards. While interesting conceptually, this is not very practical, as it forces Alice and Bob either run a lot of such key agreement protocols beforehand (defeating the desire not to pre-share a long secret), or have the server generate a fresh string X after each such agreement.

(d) *Lack of User Integrity.* While the framework of [DY21] provided some security against malicious server (or servers, in the distributed variant below), this protection was limited to privacy. Concretely, even if the server kept the randomizer X, the one-time pad $m \oplus R$ could be encrypted over a computationally secure channel. Thus, we lose everlasting privacy, but at least maintain computational privacy. On the other hand, there is no built-in integrity against malicious server, even when the computational channel has such integrity (i.e., Bob gets correct S and the ciphertext $m \oplus R$). For example, $\mathcal{S}$ could return non-matching one-time pads $R \neq R'$ to Alice and Bob, causing Bob to output a wrong message $m' = m \oplus (R \oplus R')$.

(e) *Lack of Server Authentication.* The model of [DY21] did not explicitly separate pad generation requests of Alice from the pad retrieval requests on Bob. For example, imagine that Alice sends a message to a group of people, or if Bob want to decrypt the ciphertexts multiple times. It seems reasonable that multiple retrieval of “old pads” should be allowed, and not “count” towards using up the limited randomness from the randomizer X. But it is unclear how to implement this effectively in the model of [DY21]. First, if the server is truly stateless, this seems impossible. Second, even if we allow for some small state (as we will do in our model below), it would force the server to track the number of *distinct* seed requests, which is possible (e.g., using “hyperloglog” method [HNH13]) but somewhat cumbersome and imprecise. More generally, one could imagine other situations where retrieval requests should be treated differently than generation requests (e.g., users only pay for generation, and subsequent multiple retrievals are free). For such scenarios, servers might want to have an additional security which we call *server authentication*: one should be able to only retrieve previously generated pads. Once again, the model of [DY21] does not have this property.

1.2 Our Results

We first highlight the main results of this work, which are elaborated upon in the remainder of this subsection.

- **Formalizing HELP:** We eliminate several inherent shortcomings of the existing HELP framework by introducing a relaxed, generalized syntax that allows for a short, dynamic server state. This new framework allows us to eliminate the shortcomings (a)-(e) as mentioned above.

- **Single-Server Construction:** We propose a simple single-server HELP construction that achieves nearly zero entropy waste and ensures strong security properties, including server authentication, user integrity and privacy. Our construction leverages a computational MAC scheme and introduces the notion of an *extractor-hash* for statistical randomness extraction, which we show how to build from collision-resistant hash functions.

- **General Composition:** We present one of the first successful composition theorems for everlasting privacy within the HELP framework. This enables clean, modular design of information-theoretic schemes with guaranteed everlasting privacy when combined with computationally secure channels.

- **Distributed Setting:** We generalize the single-server HELP framework to a distributed multi-server setting, significantly reducing the reliance on secure channels between servers and users. By introducing *syndrome-resilient functions* (SRFs), we achieve robust error correction and extraction properties, enabling us to extend the single-server construction to the distributed setting with near-optimal parameters. We make the same assumptions about server trustworthiness as in [DY21].

Our Formalization of HELP. In this work we eliminate these shortcoming (a)-(e). As some of the limitations are inherent in "doubly-affine extractor" restriction on the HELP framework, we do it by first relaxing the framework itself, without significantly affecting its usefulness and practicality.

Our key observation comes from the fact that the restriction on $\mathcal{S}$ to be fully stateless seems both restrictive, and likely not realizable anyway. Indeed, since Eve has direct access to the server, and the length of the randomizer X is a-priori bounded, with a stateless server everlasting security is impossible, unless Eve asks a bounded number of questions to the server. Indeed, the total number of one-time pads obtained by Bob has to be bounded by $(1-\beta)|X|$ in the framework of [DY21], where $\beta \in (0,1)$ was the "waste" parameter mentioned earlier. In practice, of course, a motivated attacker Eve can spam the server with many more requests than it is allowed. And while we can implement some indirect counter-measures (like slow sequential responses), by far the easiest solution is to allow the server to maintain some small state; at the minimum, to count the number of (distinct) requests. Moreover, since all the requests to a particular server could be serialized, it seems that allowing for short state should not be a big deal in practice.[4]

This is precisely what we do: in addition to storing a long randomizer X, which is static, we allow a short amount (say, security parameter number of bits) of state μ which the server is allowed to change. As we will see, it will immediately allow us to solve efficiency issues (a)+(b). In fact, our solutions will have almost no entropy waste, and will read $\sim |m|$ bits from X. Moreover, state allows us to not limit the number of queries Eve can make. If too many queries are made, the server simply shuts down, which in the worst case corresponds to the denial of service to legitimate users.

Once we have this, we also generalize the fact that server access should happen by necessarily sending a random seed S. (Indeed, the latter seems like the artifact of the stateless server.) And once we allow this more general syntax, we can explicitly replace the computational key-agreement step to generate S, with a more general "computational channel" allowing Alice to transmit whatever information she wants to Bob. As a side benefit, it will allow us to state and prove a much more general composition theorem in our (generalized) HELP framework, solving issue (c). Namely, like [DY21], we will define an information-theoretic version of our model (at least for privacy), which will focus on generating the one-time value R in the HELP framework (and will abstract out the message m, for now). This will roughly correspond to Alice transmitting certain values over an ideal channel to Bob. Then, we show that our scheme generically implies (formally defined) everlasting privacy (where now we introduce the message m), if the ideal channel is replaced by any computationally secure channel, formalized in the UC-Framework of [Can01], and this channel additionally transmits $m \oplus R$.

[4] This should be contrasted with trivial solutions, where the server state can grow with the number of keys, which seem much harder to implement, especially for heavy concurrent access. *Our solutions will prohibit such non-constant dynamic state.*

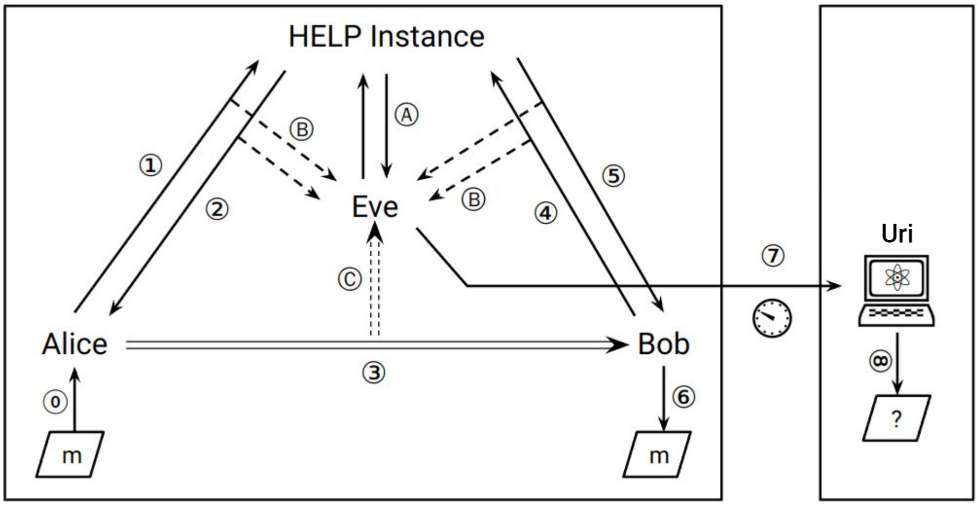
Next, to address limitation (d), we will explicitly require *user-integrity* properties against malicious HELP server(s), which will ensure that the most integrity harm caused by this server will be denial-of-service (which is inevitable). Finally, to address limitation (e), we will require *server authentication* property, which will separate pad generation requests from pad retrieval, and ensure that such retrieval is only possible for previously generated pads. As a side benefit, it will allow unbounded number of pad retrieval requests for previously generated pads. Namely, the server will only reject, if too many fresh pad generation requests are made, but retrieving old pads by legitimate users should always be possible, as long as they happen before the server erases its randomness.

The resulting framework is presented in Figure 1, where the missing notation, and other forms of "leakage" to Eve will be explained in Section 3 (for the information-theoretic part of generating R) and Section 5 (for UC-part, which also introduces the message m). Instead, we only mention some key parts: (a) everlasting privacy (and SNDL-resistance) is modeled by Eve being computationally bounded while having access to the HELP instance, and later computationally unbounded (represented by a "quantum computer" Uri on the right); (b) the computational message transmission protocol ③ could be interactive (e.g., TLS), and could happen at any time during the experiment; for example, after Eve has made many HELP queries.

Single-Server Construction. Having defined HELP syntax, we carefully define correctness and security for HELP, including server-correctness, user-correctness, server authentication, user integrity and privacy. (These properties were either undefined or only sketched by [DY21], since HELP was mainly one of the applications of their doubly-affine extractor primitive.) We then show an extremely simple single-server HELP primitive meeting our definition. In essence, the server splits X into a short key k for a computational MAC, and a long string X_{pad}. Upon request to produce n-bit pad Y, it simply reads Y from the unused portion of X_{pad}, keeping track of the starting index i of Y (which is part of its short state μ). It authenticated Y by simply tagging the corresponding tuple (i, n) with k, and returns the tag σ to Alice. Later, when Bob asks (i, n, σ) to the server, it only responds with Y if the tag value matches. By the unforgeability of the MAC, this means user can only request values previously returned by the server. In particular, the scheme is super efficient, and has (almost) no entropy waste.

The only interesting subtlety is the new *user integrity* property, protecting Alice and Bob for malicious server. Concretely, Alice wants to extract a value R from the pad Y together with the value z, so that z prevents the server from returning any $Y' \neq Y$. The naive way is to set z to be a collision-resistant hash of Y. But now we have a tricky problem of ensuring that R is statistically random even conditioned on z (which will be leaked to the unbounded attacker later). One can use a general *randomness extractor* [NZ96] to Y, since Y has a lot of entropy given z. But this requires an additional pass over Y, as well as extra seed. And is overall inelegant. Instead, we use the fact that Y is random (when the server is honest). As a result, we can set $Y = (R \| W)$, for a short suffix W, and use the iterative nature of existing hash functions to observe that $Hash(R \| W) = Hash'(Hash(Y) \| W)$. Moreover, the function $Hash'(v, W)$ is likely a *statistically hiding commitment* to v (and hence R), when W is random and sufficiently long. Hence, we can effectively do the naive thing of setting $z = Hash(Y)$, but our extractor is trivial: it takes a big prefix R or Y as its final key. More generally, in Section 4.1 we formalize the notion of *extractor-hash*, which abstracts the properties of this concrete construction, and show that such functions can be efficiently build from collision-resistant hash functions.

General Composition. We mention that there were several unsuccessful composition attempts for various everlasting privacy models. For example, the Bounded Storage Model (BSM) [Mau92, ADR02] achieves everlasting privacy assuming the attacker has limited



⓪ Alice receives a message m to send to Bob, and computes how many bits she should request, $n \leftarrow \mathsf{PadLength}(|m|)$, based on the length of m.

① Alice sends $\mathsf{Gen}(n)$ query to the HELP instance, requesting n random bits.

② The HELP instance responds with (σ, y), where σ is a tag and y is the pad.

③ Alice runs $(z, r) \leftarrow \mathsf{Auth}(\sigma, y)$ to get the randomness r and helper z. Then, Alice runs UC-secure message transmission protocol to send $(m \oplus r, \sigma, z)$ to Bob.

④ Upon receipt of (c, σ, z), Bob sends $\mathsf{Rep}(\sigma)$ query to the HELP instance.

⑤ The HELP instance responds with y.

⑥ Bob runs $r \leftarrow \mathsf{Ver}(z, y)$ and outputs the decrypted message $m = c \oplus r$.

⑦ Computationally bounded Eve outputs its view to a computationally-unbounded Uri, but only after the HELP instance becomes offline.

(∞) Uri (unsuccessfully) tries to obtain information about the message m.

Eve also has the following additional information Ⓐ, Ⓑ, Ⓒ, which will be passed to Uri:

Ⓐ Eve may interact with the HELP instance like a normal user, making Gen or Rep queries. Notice that Eve may make such queries at arbitrary times.

Ⓑ Partial leakage (compromise of several HELP servers) from the channels in steps ①, ②, ④, and ⑤.

Ⓒ Computational leakage (e.g.ciphertexts) from the UC-secure message transmission protocol during ③.

Figure 1: HELP Framework for Everlasting Privacy (we omit security parameter λ).

space, but Alice and Bob start with a short secret key S. A natural suggestion would be to replace S with a computationally secure key agreement protocol. Unfortunately, Dziembowski and Maurer [DM04] showed that such composition is flawed, by giving a convincing counter-example, and a black-box barrier to this was proved later by Harnik and Naor [HN06]. In other related settings of the streaming BSM [DQW23, GZ21] or incompressible cryptography [Dzi06, GWZ22, GWZ23], the question of building secure schemes against hybrid attackers was left open by [BCEQ24]. In general, Müller-Quade and Unruh [MU10] defined a general UC-type notion of everlasting security (termed “long-term security”), and showed that it suffers from severe non-composition issues.

Thus, it is non-trivial to have successful composition theorems for everlasting privacy. Nevertheless, in Section 5 we give one of the first such theorems, albeit in the HELP model. This shows that one can design clean and simple-to-analyze HELP schemes according to our information-theoretic (for privacy) notion in Section 3, and automatically get everlasting privacy when the idealized computation channel transmitting the tag/helper-tuple from Alice to Bob is replaced by any UC-secure message transmission (see ③ in Figure 1). The idea why this was successful was that our IT-notion has an efficiently verifiable relation under which is was clear whether the adversary broke the scheme.

Distributed Setting. Finally, in Section 7 , similar to [DY21], we generalize the single-server HELP to the distributed setting, with the goal to significantly weaken the unrealistic secure channel assumption between the server and all the users. Instead, as it is done in all paper in IT-secure multiparty computation (MPC) [BGW88], we assume that the adversary Eve can compromise at most $t_p < t$ channels between the honest user and the t servers. Additionally, we also want to protect Bob against a small number t_f servers being unavailable, and up to t_a servers giving inconsistent answers to Alice and Bob. Namely, Bob should still be able to decrypt Alice's message (or get her key R) using appropriate error-correcting techniques.

However, the simplicity of the HELP framework allows us to go much further than traditional MPC. The t servers can operate completely independently, without any knowledge about the other servers! In fact, each of these servers has the same syntax as in single-server HELP. This also means the servers never need to communicate with (or know about!) each other, and do not need any coordination about when they need to erase their randomness. Instead, each server should just independently keep it long enough for the honest users to access. And even if a small number of servers happened to accidentally erase their randomness too soon, users can overcome it by conservatively setting the value t_f corresponding to the number of “unavailable” servers.

We also use a simple coding-theory technique to effectively generalize our single-server tools to the multi-server setting. Since this technique is of independent interest, in Section 6, we abstract the resulting primitive, which we call *syndrome-resilient function* (SRF). Intuitively, SRFs simultaneously has error-correcting properties (similar to “syndrome decoding“ [HJR06]), and extraction properties akin those of *resilient functions* [CGH$^+$85]. Using SRFs, we show how our single-server construction can be extended to distributed HELP with nearly optimal parameters: roughly, to extract ℓ-bit secret, each server has to send approximately $\ell/(t - t_p - t_f - 2t_a)$ bits.

1.3 Related Work

The Bounded Storage Model [Mau92] is one of the first models for everlasting privacy, where the attacker was assumed to be space-bounded for a short period of time. Unfortunately, this beautiful model has some limitations. First, we already mentioned it cannot withstand hybrid attackers, where the initial key between Alice and Bob comes from a computationally secure key agreement protocol [DM04, HN06]. Second, space is often cheap, so it might be hard to justify the space restriction on the attacker. Motivated by this, Rabin [Rab05] semi-formally defined the *Limited Access Model* (LAM), where Alice and Bob contact a dynamically-changing sequence of servers who also continuously change their data over time. LAM could be viewed as a precursor of HELP, except the server were assumed to be ad hoc, had possibly non-random data, and not guaranteed to erase their data in a timely manner (the inspiration came from the World-Wide Web, where it is hard to monitor giant amount of constantly changing data). In contrast, the HELP servers are specifically designed to help users derive randomness, and have well defined syntax, security and data erasure policies, making this model more scalable, and amendable to formal analysis.

Another related model is that of (Perfectly) *Secure Message Transmission* (SMT), pioneered by [DDWY93]. In this model there are several "disjoint communication paths" between Alice and Bob, and the attacker is assumed to monitor a bounded number of such paths. While similar to our distributed HELP modeling, SMT requires that Bob is on-line when Alice sent her message. Which is inconvenient, as Alice and Bob need to a-priori agree on which channels to use, but also rules out application where Bob is "Alice in the future".[5] Moreover, the on-line assumption makes it easier to perform a coordinated SNDL attack, as compared to our setting.

Yet another related topic is Quantum Key Distribution (QKD) [BB14, Ren08], which distributes an information-theoretically secure key through quantum channels. Specifically, there is a line of work [Ell02, BPP05, ARML06, LBD07b, LBD07a, LBD08, ABB+14, CZL+21, FYLW+22, BZG+23, VM24, MW24] in QKD that achieves this by utilizing a number of trusted, untrusted or semi-trusted QKD relays. Furthermore, the syndrome-resilient functions developed in Section 6 of this work, which address both error correction and randomness extraction, bear strong connections to the error-correction/reconciliation and privacy amplification phases of QKD. However, while the QKD setting also achieves information-theoretic security (assuming authenticated channels), the HELP setting studied in this paper is purely classical. This means all our results can be implemented today, and do not require the power of quantum computers.

Finally, our new technical tools of extractor-hash and syndrome-resilient functions, can be viewed as highly optimized special cases of randomness extractors [NZ96] and fuzzy extractors [DORS08], respectively, where one starts with the uniform distribution, and knows precisely the type of "leakage" one needs to withstand.

Paper Organization. The rest of the paper is organized as follows. In Section 2, we present the relevant preliminaries for the paper. In Section 3, we define HELP for the single server setting. In Section 4, we construct HELP that satisfies our definition (along the way defining and constructing *extractor-hash*). Then in Section 5, we show how to compose the HELP instance with a UC-secure message transmission scheme to obtain everlasting privacy. In Section 6, we define and construct Syndrome Resilient Function, which we use to construct HELP in the distributed setting. The actual distributed HELP definition and construction are discussed in Section 7.

2 Preliminaries

For a natural number $n \in \mathbb{N}$, we use the notation $[n] = \{1, \dots, n\}$. In general, lower-case letters will represent values and vectors of values, while upper-case letters will represent random variables — the exception to this is the servers' internal randomness, which we denote by X even after it is sampled. A tilde over a letter will usually represent a recovered value whose correctness may not be trusted. For universe U, we will denote by $x \sim U$ or

[5] Alternatively, each channel can buffer the message until Bob comes on-line, making its storage grow with the number of uses, and would require some kind of user authentication solution. Both of these deficiencies are not needed in our model.

$x \xleftarrow{\$} U$ drawing x from U uniformly at random. PPT stands for "Probabilistic Polynomial Time".

Definition 1 (Negligible Function)**.** A function $f : \mathbb{N} \to \mathbb{R}$ is *negligible*, denoted $\texttt{negl}(n)$, if for all $c > 0$, there exists a value n_0 such that for every $n > n_0$, $|f(n)| < 1/n^c$.

Definition 2 (Statistical Distance)**.** Let D_1 and D_2 be two distributions with support in X. The statistical distance between D_1 and D_2 is

$$\Delta(D_1, D_2) = \frac{1}{2} \sum_{x \in X} |\Pr[D_1 = x] - \Pr[D_2 = x]|.$$

We will later use the following Lemma in the proof for Theorem 5.

Lemma 1. *Let A_0, A_1, B_0, B_1 be random boolean variables, then we have*

$$|\Pr[A_0] - \Pr[A_1]| \leq \Pr[\neg B_0] + \Pr[\neg B_1] + |\Pr[A_0 \wedge B_0] - \Pr[A_1 \wedge B_1]|.$$

Proof.

$$\begin{aligned} |\Pr[A_0] - \Pr[A_1]| &= |\Pr[A_0 \wedge B_0] + \Pr[A_0 \wedge \neg B_0] - \Pr[A_1 \wedge B_1] - \Pr[A_1 \wedge \neg B_1]| \\ &\leq |\Pr[A_0 \wedge \neg B_0] - \Pr[A_1 \wedge \neg B_1]| + |\Pr[A_0 \wedge B_0] - \Pr[A_1 \wedge B_1]| \\ &\leq \Pr[A_0 \wedge \neg B_0] + \Pr[A_1 \wedge \neg B_1] + |\Pr[A_0 \wedge B_0] - \Pr[A_1 \wedge B_1]| \\ &\leq \Pr[\neg B_0] + \Pr[\neg B_1] + |\Pr[A_0 \wedge B_0] - \Pr[A_1 \wedge B_1]| \end{aligned}$$

□

This work is concerned with authenticating randomness to users querying a server. We will use message authentication codes (MACs) throughout. Our primary tool in particular are computational MACs, which we define below.

Definition 3 (Message Authentication Code)**.** Let λ be a security parameter, and let $\ell_{\texttt{MAC}}, \ell_{\texttt{tag}} \in \mathbb{N}$. Let $(\mathsf{Mac.Gen}, \mathsf{Mac.Tag}, \mathsf{Mac.Ver})$ be a tuple of algorithms with the following syntax:

- $\mathsf{Mac.Gen}(1^\lambda) \to k$: takes in the security parameter and outputs a MAC key k.
- $\mathsf{Mac.Tag}_k(m) \to t$: takes in a MAC key k and a message m of maximum length $\ell_{\texttt{MAC}}$ and outputs an associated tag t of length $\ell_{\texttt{tag}}$.
- $\mathsf{Mac.Ver}(k, m, t) \to 0/1$: verifies whether the tag t is a valid tag on the message m using the MAC key k.

We say that $(\mathsf{Mac.Gen}, \mathsf{Mac.Tag}, \mathsf{Mac.Ver})$ is a (computational) *message authentication code* if the following are true:

1. *Correctness:* For all messages m and keys k, we have

$$\Pr[\mathsf{Mac.Ver}(k, m, \mathsf{Mac.Tag}_k(m)) = 1] = 1.$$

2. *Unforgeability:* For all PPT $\mathcal{A}$, there exists a negligible function ε such that

$$\Pr\left[\mathsf{Mac.Ver}(k, m, t) = 1 \wedge m \notin \{m_1, \ldots, m_q\} \;\middle|\; \begin{array}{l} k \leftarrow \mathsf{Mac.Gen}(1^\lambda), \\ (m, t) \leftarrow \mathcal{A}^{\mathsf{Mac.Tag}_k}(1^\lambda) \end{array}\right] \leq \varepsilon,$$

where $m_1, \ldots, m_q$ are the queries made by the adversary to $\mathsf{Mac.Tag}_k$.

In many cases, $\mathsf{Mac.Gen}$ will be simply to sample a random key from a set (such as binary strings of the key length). In addition, for many MACs, the verification algorithm $\mathsf{Mac.Ver}$ is simply to run the MAC with the input and check that it matches the tag (i.e., $\mathsf{Mac.Ver}(k, m, t) = 1$ if and only if $\mathsf{Mac.Tag}_k(m) = t$). For ease of reading, we will restrict our view to such cases through the rest of the paper. In this case, we may simply describe the signing algorithm $\mathsf{Mac.Tag}$ as the MAC.

While we will primarily prove our results using a computational MAC, we note our constructions can also be instantiated using one-time message authentication. Below, we provide the unforgeability for one-time MACs.

Definition 4 (One-Time MAC Unforgeability)**.** Let λ be the security parameter, $\ell_{\mathtt{MAC}}, \ell_{\mathtt{tag}} \in \mathbb{N}$, and let $(\mathsf{Mac.Gen}, \mathsf{Mac.Tag}, \mathsf{Mac.Ver})$ be algorithms with syntax above. We say that $(\mathsf{Mac.Gen}, \mathsf{Mac.Tag}, \mathsf{Mac.Ver})$ is a *one-time MAC* if it satisfies MAC correctness and for all (computationally unbounded) $\mathcal{A}$, there exists some $\varepsilon = \mathtt{negl}(\lambda)$ such that

$$\Pr\left[\mathsf{Mac.Ver}(k, m', t) = 1 \wedge m' \neq m \;\middle|\; \begin{array}{l} k \leftarrow \mathsf{Mac.Gen}(1^\lambda), m \leftarrow \mathcal{A}(1^\lambda); \\ (m', t) \leftarrow \mathcal{A}(\mathsf{Mac.Tag}_k(m)) \end{array}\right] \leq \varepsilon.$$

One-time MACs have the advantage of being more light weight and information theoretic (from, e.g., pairwise independent hashing) at the cost of needing to resample a key each time. In our applications, this will involve the server having to store more internal randomness for each query (as only a single key will need to be stored for all queries in the computational MAC variant). As such, we present our results with computational MACs and include discussion on the differences in parameters when relevant.

3 Defining Single Server HELP

3.1 Syntax and Correctness

The HELP scheme consists of a tuple $(\mathsf{Init}, \mathsf{PadLength}, \mathsf{Auth}, \mathsf{Ver}, \mathsf{Gen}, \mathsf{Rep})$ of algorithms, where $(\mathsf{Init}, \mathsf{Gen}, \mathsf{Rep})$ are run by the server, while $(\mathsf{PadLength}, \mathsf{Auth}, \mathsf{Ver})$ are run by the users.

Syntax. The server-based algorithms have the following syntax:

- $\mathsf{Init}(N, 1^\lambda) \to (X, \mu)$. The initialization algorithm run by the server $\mathcal{S}$, where:
  - N is the total length of all pads ever requested by the users, and λ is the security parameter .
  - X is a truly random string of some length $|X| = \tilde{N} \geq N$, which will stay static throughout the life-time of the system.
  - μ is the initial state of the server, which is meant to be compact, but can vary by future calls to Gen (but not Rep; see below).
- $\mathsf{Gen}(n, (X, \mu)) \to (\sigma, y, \mu)$. Pad generation algorithm run by server $\mathcal{S}(X, \mu)$, which returns a sample y of length $|y| = n$, its tag σ, and updates the server state μ.
- $\mathsf{Rep}(\tilde{\sigma}, (X, \mu)) \to \tilde{y}$. Pad reproduction algorithm run by the server $\mathcal{S}(X, \mu)$, which takes tag $\tilde{\sigma}$, and returns the pad $\tilde{y}$ corresponding to $\tilde{\sigma}$.

The user-based algorithms have the following syntax:

- $\mathsf{PadLength}(\ell, 1^\lambda) \to n$. Length calculation algorithm run by the user A, which outputs how many bits n the user should request from $\mathcal{S}$, if A wants to derive an ℓ-bit key.

- $\mathsf{Auth}(\sigma, y, 1^\lambda) \to (z, r)$. Transforms n-bit pad y returned by S into an ℓ-bit key r, and helper z.
- $\mathsf{Ver}(z, \tilde{y}) \to \tilde{r}$. Checks helper z against pad $\tilde{y}$, and outputs a key $\tilde{r} \in \{0,1\}^\ell \cup \{\perp\}$.

Correctness. We split the correctness into two parts. First, *server correctness* states that calls to Gen and Rep always return the same value, if Rep uses the tag σ returned by Gen. More formally,

Definition 5. Let $\lambda, N \in \mathbb{N}$. Assume $n_1, \ldots, n_q$ are integers such that $\sum_{i=1}^q n_i \le N$. Let us denote the initial state of the server $(X, \mu_0) \leftarrow \mathsf{Init}(N, 1^\lambda)$. Then HELP satisfies *server-correctness* if, for all $1 \le i \le q$, we have:

$$\Pr\left[\ \tilde{y}_i = y_i \ \middle|\ (\sigma_i, y_i, \mu_i) \leftarrow \mathsf{Gen}(n_i, (X, \mu_{i-1})), \tilde{y}_i \leftarrow \mathsf{Rep}(\sigma_i, (X, \mu_i))\ \right] = 1.$$

Similarly, for *user-correctness* we require that the value R produced by Auth is always recovered by Ver when used with the correct value z. More formally,

Definition 6. HELP satisfies *user-correctness* if, for all $y \in \{0,1\}^n$, and all σ we have:

$$\Pr\left[\ \tilde{r} = r \ \middle|\ (z, r) \leftarrow \mathsf{Auth}(\sigma, y, 1^\lambda),\quad \tilde{r} \leftarrow \mathsf{Ver}(z, y)\ \right] = 1.$$

It is immediately clear that server-correctness and user-correctness imply the overall correctness of the HELP scheme, which states that users should recover the same keys by Auth and Ver when using correct tag and helper values $\tilde{\sigma} = \sigma$ and $\tilde{z} = z$.

Definition 7. Let $\lambda, L \in \mathbb{N}$. Assume $\ell_1, \ldots, \ell_q$ are integers such that $\sum_{i=1}^q \ell_i \le L$, $n_i = \mathsf{PadLength}(\ell_i, 1^\lambda)$, and N is an integer such that $\sum_{i=1}^q n_i \le N$. Let us denote the initial state of the server $(X, \mu_0) \leftarrow \mathsf{Init}(N, 1^\lambda)$. Then HELP satisfies server-aided *correctness* if, for all for all $1 \le i \le q$, we have:

$$\Pr\left[\ \tilde{r}_i = r_i \ \middle|\ \begin{array}{c} (\sigma_i, y_i, \mu_i) \leftarrow \mathsf{Gen}(n_i, (X, \mu_{i-1})), (z_i, r_i) \leftarrow \mathsf{Auth}(\sigma_i, y_i, 1^\lambda), \\ \tilde{y}_i \leftarrow \mathsf{Rep}(\sigma_i, (X, \mu_i)), \qquad \tilde{r}_i \leftarrow \mathsf{Ver}(z_i, \tilde{y}_i) \end{array} \right] = 1.$$

Before proceeding to security, we will make a notational convention, which will make our definition easier to parse. Namely, we will often omit explicit reference to server state (X, μ), and simply write $\mathsf{Gen}(n) \to (\sigma, y)$ and $\mathsf{Rep}(\tilde{\sigma}) \to \tilde{y}$. With the understanding that:

(1) X always stays the same;

(2) μ is correctly updated by every call to Gen.

For example, the Correctness condition in Definition 7 becomes easier to parse with this convention:

$$\Pr\left[\ \tilde{r}_i = r_i \ \middle|\ \begin{array}{c} (\sigma_i, y_i) \leftarrow \mathsf{Gen}(n_i), (z_i, r_i) \leftarrow \mathsf{Auth}(\sigma_i, y_i, 1^\lambda), \\ \tilde{y}_i \leftarrow \mathsf{Rep}(\sigma_i), \qquad \tilde{r}_i \leftarrow \mathsf{Ver}(z_i, \tilde{y}_i) \end{array} \right] = 1.$$

Further, in some of our notions we will give the attacker various oracles related to Gen and Rep procedures. As we are omitting explicit reference to the server's state (X, μ), it should be understood that in these oracles the attacker can only supply "user-specified" inputs, but not any part of the server's state (X, μ). For example, in the $\mathsf{Gen}(\cdot)$ oracle the attacker can only choose the pad length n (but the state μ will update after the call), and in the $\mathsf{Rep}(\cdot)$ oracle the attacker can only choose a tag $\tilde{\sigma}$.

Remark 1. In most natural schemes, $\mathsf{Auth}()$ will sample some "reusable object" h (such as a hash function), which can be safely used by future calls to $\mathsf{Auth}()$. To minimize the number of algorithms, in the current formalization the $\mathsf{Auth}()$ algorithm will sample a fresh value of h for each call, and include it in the helper value z. In practice, we expect the "reusable part" h will be sampled only once, and not included in the value of z. Both versions are equally secure, but the second one is obviously preferable. When clear from the context, we will slightly abuse the notation, and not include the "reusable part" h in the helper value z.

3.2 Security

The security of $\mathsf{HELP} = (\mathsf{Init}, \mathsf{PadLength}, \mathsf{Auth}, \mathsf{Ver}, \mathsf{Gen}, \mathsf{Rep})$ will consist of three components:

(a) *Server Authentication*, protecting the server from malicious users;

(b) *User Integrity*, protecting honest users from malicious server; and

(c) *Privacy*, protecting honest users from eavesdroppers who have (partial) access to honest server.

SERVER AUTHENTICATION. Intuitively, the server $\mathcal{S}$ wants to ensure that the only way to call the reproduction function $\mathsf{Rep}(\sigma^*)$ successfully, is to use some value σ^* returned by a previous call to Gen.

Definition 8. Let λ be the security parameter. We say that HELP satisfies {bounded, unbounded} *ε-server authentication* for $\varepsilon = \mathtt{negl}(\lambda)$ if, for all $\{PPT, \textit{unbounded}\}$ $\mathcal{A}$ with oracle access to $\mathsf{Gen}, \mathsf{Rep}, \mathsf{Gen}^*$, and any value of N, after we run $\mathsf{Init}(N, 1^\lambda)$ to initialize (X, μ), we have

$$\Pr\left[\ \mathsf{Rep}(\sigma^*) \neq \perp \wedge\ \sigma^* \notin \{\sigma_1, \ldots, \sigma_q\}\ \middle|\ \sigma^* \leftarrow \mathcal{A}^{\mathsf{Gen},\mathsf{Rep},\mathsf{Gen}^*}(N, 1^\lambda)\ \right] \leq \varepsilon,$$

where $\mathsf{Gen}, \mathsf{Rep}$ are defined as before, $\mathsf{Gen}^*(n)$ calls $(\sigma, y) \leftarrow \mathsf{Gen}(n)$ but only returns y, and $\{\sigma_1, \ldots, \sigma_q\}$ denotes the set of tags returned in response to the adversary's queries to Gen.

Note, calls to Gen^* are used to model queries made to Gen by other honest users, where the attacker is not allowed to see the tag σ returned by such a call (but might be able to get some information about the pad y by other means).[6] Moreover, we allow $\mathcal{A}$'s forged tag σ^* to be equal to such "erased" tag σ, meaning it should be hard for the attacker to compute any such "erased" tag. Because doing so will make the attacker successful in breaking the server authentication game, by outputting $\sigma^* = \sigma$. Thus, the attacker can only succeed in making a Rep call, by explicitly using a tag σ returned by a prior call to Gen.

Looking ahead, also notice that we will be able to satisfy bounded server authentication with $\tilde{N} = |X| \approx N$, while for unbounded authentication we will use slightly longer $\tilde{N} = |X| \approx O(N)$. Interestingly, when compiling our notion to a more realistic server authentication notion in Section 5.3, we will always end up with computational security, irrespective on whether we start with bounded or unbounded server authentication. Thus,

[6] While in the current definition the attacker does not get any information about tags σ produced by Gen^* calls, in Section 5.3 we will make a more realistic definition where the attacker gets some "computational information" about such tags. And also show how our current definition of server authentication implies this more realistic definition.

in practice there might be little reason to strive for unbounded server authentication, other than minimizing complexity assumptions.

User Integrity. Notice, in our main application, the user performing authorization $\mathsf{Auth}(\sigma, Y) \to (z, r)$ might either be different from the user calling $\mathsf{Ver}(\tilde{z}, \tilde{y}) \to \tilde{r}$, or otherwise not have the correct values $\tilde{z} = z$ and $\tilde{y} = y$. In such cases there could be a real danger for the $\tilde{r} \neq r$. In particular, a malicious server could potentially return inconsistent values y and $\tilde{y}$ on the two corresponding calls to Gen and Rep. User integrity ensures that such malicious server is limited to the denial of service attack, *provided that the helper values are correct* (i.e., $\tilde{z} = z$). This means that, as long as the server $\mathcal{S}$ is computationally bounded, either $\tilde{r} = r$ (correctness still holds), or $\tilde{r} = \perp$ (denial of service).

Thus, as long as authenticity of the helper value z is ensured, server cannot cause the user to output inconsistent keys. Put differently, authenticating the helper value z is implicitly authenticating the derived key r. Point being that authenticating z is easier than r for our application, as z does not need to be secret, while we aim for r to be everlastingly private.

Definition 9. Let λ be the security parameter. We say that HELP satisfies *δ-user integrity* with $\delta = \mathtt{negl}(\lambda)$, if for all PPT servers $\mathcal{S}$, we have

$$\Pr\left[\ \tilde{r} \notin \{r, \perp\}\ \middle|\ \begin{array}{c} (st, \sigma, y) \leftarrow \mathcal{S}(1^\lambda); (z, r) \leftarrow \mathsf{Auth}(\sigma, y, 1^\lambda); \\ \tilde{y} \leftarrow \mathcal{S}(st, z, r); \quad \tilde{r} = \mathsf{Ver}(z, \tilde{y}) \end{array}\ \right] \leq \delta$$

Note, in this game we do not even require $\mathcal{S}$ to maintain any (random) database X, or follow any rules that the honest server would follow.

Privacy. Now we define our notion of privacy for a HELP scheme. Intuitively, it states that randomness R generated with the help of the honest server is *unconditionally secure*, even against attacker who made many pad generation queries Gen to the server, and knows the authentication values σ and z associated with R. Notice, such (surprisingly) strong security is only possible because the attacker is not given access to the pad reproduction oracle Rep, which would have trivially rendered this notion impossible. Nevertheless, we will later lift this restriction in Section 5.5, where we will show that our notion of Privacy, coupled with Server Authentication,[7] will imply the notion of *Everlasting Privacy* discussed in the Introduction. Where the attacker is initially computationally bounded and has access to Rep, but can become unbounded after losing access to Rep, and potentially learning the challenge values (σ, z).

Definition 10. Let λ be the security parameter. We say that HELP satisfies *ξ-privacy* for $\xi = \mathtt{negl}(\lambda)$, if for all *unbounded* adversaries $\mathcal{A}$ with oracle access to Gen, and any value of N, after we run $\mathsf{Init}(N, 1^\lambda)$ to initialize (X, μ), and sample a random bit $b \overset{\$}{\leftarrow} \{0,1\}$, we have

$$\Pr\left[\ b' = b\ \middle|\ \begin{array}{l} (\ell, st) \leftarrow \mathcal{A}^{\mathsf{Gen}}(N, 1^\lambda); n \leftarrow \mathsf{PadLength}(\ell, 1^\lambda); \\ (\sigma, y) \leftarrow \mathsf{Gen}(n); (z, r_0) \leftarrow \mathsf{Auth}(\sigma, y, 1^\lambda); \\ r_1 \overset{\$}{\leftarrow} \{0,1\}^\ell; b' \leftarrow \mathcal{A}^{\mathsf{Gen}}(st, \sigma, z, r_b) \end{array}\ \right] \leq \frac{1}{2} + \xi.$$

4 Constructing Single Server HELP

We will now construct single server-aided HELP from message authentication codes (MAC) and collision-resistant hash functions (CRHF).

[7] Plus a “message transmission functionality” which we will define later.

4.1 Extractor-Hash

In order to realize HELP in the single server setting, we introduce the new notion of an *extractor-hash*. Intuitively, when the user gets the pad value y from the server, she has to extract a random key r and the helper value z, s.t.: (a) z is commitment to r; but (b) r is information-theoretically secure given z. This is easy to do theoretically, by setting z to be a collision-resistant hash function (CRHF) h applied to y, and then extracting randomness r from the source y conditioned on "leakage" z. While this approach works, it is practically inefficient for two reasons. First, this requires scanning the input y twice — once for hashing, and once for extracting. Second, provably secure randomness extractors (e.g., given by the Leftover Hash Lemma [ILL89]) require an extra random seed, and have non-trivial entropy loss. Instead, by abstracting the security properties of our extractor-hash primitive, we take advantage of the fact that the initial source y is truly random, and achieve a much more efficient solution. Details follow.

For security parameter $\lambda \in \mathbb{N}$, let $\mathcal{E}_\lambda = \{\mathsf{EH} : \{0,1\}^* \to \{0,1\}^{\ell_{\mathtt{help}}} \times \{0,1\}^*\}$ be a family of efficient functions, where $\ell_{\mathtt{help}} = \ell_{\mathtt{help}}(\lambda)$, and let $\mathsf{EH} \leftarrow \mathcal{E}_\lambda$ be a randomly chosen member of such a family. Given $y \in \{0,1\}^n$, we denote the outputs of EH by $(z,r) = \mathsf{EH}(y)$, where $z \in \{0,1\}^{\ell_{\mathtt{help}}}$ and $r \in \{0,1\}^\ell$ for some $\ell = \ell(n,\lambda)$. We call the helper length $\ell_{\mathtt{help}}$ the *compactness* of $\mathcal{E}_\lambda$ and the value $(n-\ell)$ as the *entropy loss* of $\mathcal{E}_\lambda$.

Definition 11. The family $\mathcal{E}_\lambda = \{\mathsf{EH} : \{0,1\}^* \to \{0,1\}^{\ell_{\mathtt{help}}} \times \{0,1\}^*\}$ is an *extractor-hash family* if for a randomly chosen $\mathsf{EH} \leftarrow \mathcal{E}_\lambda$, we have:

1. *Collision-Resistance.* For all PPT adversaries $\mathcal{A}$,

$$\Pr\left[\, z_1 = z_2 \wedge r_1 \neq r_2 \;\middle|\; \begin{array}{l} (y_1,y_2) \leftarrow \mathcal{A}(\mathsf{EH},1^\lambda); \\ (z_1,r_1) = \mathsf{EH}(y_1), (z_2,r_2) = \mathsf{EH}(y_2) \end{array} \right] \leq \mathtt{negl}(\lambda).$$

2. *Extraction.* There exists some function $\delta = \mathtt{negl}(\lambda)$ such that, for any n and a randomly sampled $Y \xleftarrow{\$} \{0,1\}^n$, if $(Z,R) = \mathsf{EH}(Y)$, then

$$\Delta((Z,R),(Z,U_\ell)) \leq \delta,$$

where $\ell = |R|$, and U_ℓ denotes the uniform distribution of ℓ-bit strings.

Notice that the first property is exactly collision resistance on the extracted key r, while the second is a statistical hiding property on the extracted key r. Moreover, we only need the extraction property to hold for a randomly sampled value y, which will allow for a super-efficient construction below.

Remark. Similar to collision-resistant hash function families, we define a family of function for extractor-hashes. In our application to HELP, we will want to select one of these uniformly from the set $\mathcal{E}_\lambda$ and use it throughout. One way to ensure this is to add the description of EH to $\mathsf{Auth}, \mathsf{Ver}$. Users could also agree upon the choice of $\mathsf{EH} \leftarrow \mathcal{E}_\lambda$ in a preprocessing step. We will not write either of these choices explicitly, though we note here that neither poses any issue to the above properties of EH.

Construction. We will now show how to construct an extractor-hash from any family of collision-resistant hash functions (CRHF) and any statistically hiding commitment (SHC) scheme on short messages, which we define below.

Definition 12 (Collision-Resistant Hash Function (CRHF))**.** We say $\mathcal{H}_\lambda = \{h : \{0,1\}^* \to \{0,1\}^{\ell_{\mathtt{hash}}}\}$ is a family of collision-resistant hash functions with output length $\ell_{\mathtt{hash}} = \ell_{\mathtt{hash}}(\lambda)$, if for a randomly sampled $h \leftarrow \mathcal{H}_\lambda$ and for every PPT $\mathcal{A}$, we have

$$\Pr[\; h(x) = h(x') \;\wedge\; x \neq x' \;\mid\; (x,x') \leftarrow \mathcal{A}(h,1^\lambda) \;] \leq \mathtt{negl}(\lambda).$$

Next we define the notion of SHCs. Notice, we only needs SHCs on short messages (of length $\ell_{\mathtt{hash}}$, which is the CHRF output).

Definition 13 (Statistically Hiding Commitments)**.** We say $\mathcal{C}_\lambda = \{\mathsf{Com} : \{0,1\}^{\ell_{\mathtt{hash}}} \times \{0,1\}^{\ell_{\mathtt{rand}}} \to \{0,1\}^{\ell_{\mathtt{SHC}}}\}$ is a family of *statistically-hiding commitments* (SHCs) for input length $\ell_{\mathtt{hash}} = \ell_{\mathtt{hash}}(\lambda)$ and randomness length $\ell_{\mathtt{rand}} = \ell_{\mathtt{rand}}(\lambda)$ if for a randomly chosen $\mathsf{Com} \leftarrow \mathcal{C}_\lambda$, we have the following two properties:

1. **Computationally Binding:** For all PPT $\mathcal{A}$, we have
$$\Pr\left[\mathsf{Com}(v;w) = \mathsf{Com}(v';w') \wedge v \neq v' \mid (w',w,v',v) \leftarrow \mathcal{A}(\mathsf{Com},1^\lambda)\right] \leq \mathtt{negl}(\lambda).$$
2. **Statistically Hiding:** There exists $\delta = \mathtt{negl}(\lambda)$ such that for any messages $v, v' \in \{0,1\}^{\ell_{\mathtt{hash}}}$, we have
$$\Delta(\mathsf{Com}(v;U_{\ell_{\mathtt{rand}}}), \mathsf{Com}(v';U_{\ell_{\mathtt{rand}}})) \leq \delta$$

With these in mind, we present our construction.

Construction 4.1. *Let $\mathcal{H} = \{h\}$ be a CRHF with output length $\ell_{\mathtt{hash}}$, and let $\mathcal{C} = \{\mathsf{Com}\}$ be a SHC for input length $\ell_{\mathtt{hash}}$ and randomness length $\ell_{\mathtt{rand}}$. Then these families define a family $\mathcal{E} = \{\mathsf{EH} : \{0,1\}^* \to \{0,1\}^{\ell_{\mathtt{SHC}}} \times \{0,1\}^*\}$ with compactness $\ell_{\mathtt{SHC}}$ and entropy loss $\ell_{\mathtt{rand}}$, as follows.*

- *Given input $y \in \{0,1\}^n$, parse it as $y = (r||w)$, where $w \in \{0,1\}^{\ell_{\mathtt{rand}}}$ and $r \in \{0,1\}^\ell$ with $\ell = n - \ell_{\mathtt{rand}}$. Then define helper string $z = \mathsf{Com}(h(r);w)$, and output*
$$\mathsf{EH}(y) := (z, r)$$

Theorem 1. *Assuming CRHF and SHC satisfy Definitions 12 and 13, respectively, then Construction 4.1 is a secure Extractor-Hash family.*

Proof. To prove collision-resistance, suppose we have an efficient algorithm $\mathcal{A}$ that outputs (with non-negligible probability) y_1, y_2, for which $r_1 \neq r_2$, but $z_1 = z_2$, where $(z_1, r_1) = \mathsf{EH}(y_1)$ and $(z_2, r_2) = \mathsf{EH}(y_2)$. Since $z_i = \mathsf{Com}(h(r_i);w_i)$, the binding property of SHCs implies that we must have $h(r_1) = h(r_2)$. But then the collision-resistance of h further implies $r_1 = r_2$, which is a contradiction to our assumption. In short, if such an $\mathcal{A}$ exists, it would break either the binding property of the SHC, or the collision-resistance of h.

To show the extraction property, notice that when $Y = (R, W)$ is truly random, R by itself is perfectly random, and independent of W. Let us sample another value $R' \leftarrow U_\ell$. For any particular fixing of values $(r, r') \sim (R, R')$, which in turn fixes values $v = h(r)$ and $v' = h(r')$, statistically hiding property of SHCs on (v, v') implies that
$$\Delta(\mathsf{Com}(h(r);W), (\mathsf{Com}(h(r');W)) \leq \delta = \mathtt{negl}(\lambda)$$
Taking the average over (R, R'), we then get
$$\Delta((\mathsf{Com}(h(R);W), R, R'), (\mathsf{Com}(h(R');W), R, R')) \leq \delta$$
Applying a truncation of R' operation to both sides, we get
$$\Delta((\mathsf{Com}(h(R);W), R), (\mathsf{Com}(h(R');W), R)) \leq \delta$$
But now we rename R and R' on the right-hand side, and get
$$\Delta((\mathsf{Com}(h(R);W), R), (\mathsf{Com}(h(R);W), R')) \leq \delta$$
which is exactly $\Delta((Z, R), (Z, R')) \leq \delta$ we needed, since $Z = \mathsf{Com}(h(R);W)$. □

Instantiating Extractor-Hash. Notice, the existence of CRHFs with output length $O(\lambda)$ implies the existence of SHCs [DPP94] with randomness $O(\lambda)$ for committing to $O(\lambda)$-bit length messages. Thus, we get:

Corollary 1. *Assuming the existence of CRHFs with output length $O(\lambda)$, there exists an efficient Extractor-Hash family with compactness and entropy loss $O(\lambda)$.*

In practice, however, one can instantiate our construction even more efficiently, via a single call to an existing CHRF, such as SHA-2, SHA-3, or SHA-256.

That is, consider a CRHF which is an iterative process like the Merkle-Dåmgard transform [Dam89] applied to an appropriate compression function h, so that hashing an input $x = (x_1, x_2, \ldots, x_n)$ of n blocks is represented by iteratively hashing $h(h(\ldots(h(h(0^n, x_1), x_2), \ldots), x_n)$, possibly with some other finalization process. In Construction 4.1, we would then apply some CRHF-based SHC to the output of this process. If we instead are able to:

1. Pull out a finalizing function Finalize from the hash which heuristically satisfies the properties of SHC (say, the last hash or two of the above Merkle-Dåmgard-based process) when x_n is random, and

2. Show the first steps of the hash when this finalization is removed is still collision-resistant,

we could simplify our construction of EH to be just a single call to the iterative hash function, say SHA-2. This would in effect save us the cost of running this finalization procedure twice.

We show this is feasible when the CRHF used is of the SHA family (i.e., SHA-2, SHA-3, SHA-256, SHA-512). In particular, the first property is satisfied if the function Finalize is collision-resistant and its output are close to uniform. If we assume these, though, then we see from prior work that the second property follows for the SHA family of hash functions. This follows from Corollary 1 and Lemma 2 of [DP08], which state that if h is collision resistant and its outputs are regular, then the overall Merkle-Dåmgard composition is also collision resistant, as well as Merkle-Dåmgard with truncation on the compression function. So, using a heuristic assumption that a (few Merkle-Dåmgard rounds of) SHA compression function is close to random, and assuming the inputs to the EH are variable length, we see that a single call to a SHA hash is sufficient for an extractor-hash.

4.2 Main Single-Server Scheme

Construction 4.2. *For security parameter λ, $\mathcal{E}_\lambda = \{\mathsf{EH} : \{0,1\}^* \to \{0,1\}^{\ell_{\texttt{help}}} \times \{0,1\}^*\}$ an extractor-hash, and let $\mathsf{Mac.Tag} : \{0,1\}^* \times \{0,1\}^{\ell_{\texttt{MAC}}} \to \{0,1\}^{\ell_{\texttt{tag}}}$ be a computational MAC. We define $\mathsf{HELP} = (\mathsf{Init}, \mathsf{Gen}, \mathsf{Rep}, \mathsf{PadLength}, \mathsf{Auth}, \mathsf{Ver})$ as so, starting first with the initialization and length:*

- $\mathsf{Init}(N, 1^\lambda)$*: On input N, set $\tilde{N} = N + \ell_{\texttt{MAC}}$ and sample $X \leftarrow \{0,1\}^{\tilde{N}}$ uniformly at random. Parse $X = (k, X_{pad})$, where $|k| = \ell_{\texttt{MAC}}$. Set $\mu = (1, N)$.*

- $\mathsf{PadLength}(\ell, 1^\lambda)$*: On input $(\ell, 1^\lambda)$, output $n = \ell + \ell_{\texttt{help}}$.*

With these in mind, we can now define how the server accepts and responds to Gen, Rep *queries:*

- $\mathsf{Gen}(n)$*: Let $\mu = (\mathsf{index}, N)$. If $\mathsf{index} + n > N$, return $\perp$. Otherwise, define $y = X_{pad}[\mathsf{index}, \ldots, \mathsf{index} + n - 1]$. Set $\sigma = (\mathsf{Mac.Tag}_k(\mathsf{index}, n), \mathsf{index}, n)$. Then, output (y, σ) and set $\mu = (\mathsf{index} + n, N)$.*

- $\mathsf{Rep}(\sigma)$*: Parse* $\sigma = (t, \mathit{index}, n)$*. Then, if* $t = \mathsf{Mac.Tag}_k(\mathit{index}, n)$*, return* $\tilde{y} = X_{pad}[\mathit{index}, \ldots, \mathit{index} + n - 1]$*. Otherwise, return* $\tilde{y} = \perp$*.*

Finally, to parse these queries for pads, we define:

- $\mathsf{Auth}(\sigma, y)$ *: On input* (σ, y)*, output* $(z, r) \leftarrow \mathsf{EH}(y)$*.*
- $\mathsf{Ver}(z, \tilde{y})$*: On input* $(z, \tilde{y})$*, set* $(\tilde{z}, \tilde{r}) \leftarrow \mathsf{EH}(\tilde{y})$*. If* $z = \tilde{z}$*, output* $\tilde{r}$*. Otherwise, output* $\perp$*.*

Theorem 2. *For security parameter* λ*, if* $\mathcal{E}_\lambda$ *is an extractor-hash and* $\mathsf{Mac.Tag}$ *is a computational MAC, then Construction 4.2 is a* HELP*. Under standard assumptions about CHRFs and MACs, this gives nearly optimal randomizer length* $\tilde{N} = N + O(\lambda)$*, and server overhead* $n = \ell + O(\lambda)$ *per* ℓ*-bit extraction.*

Proof. Correctness follows from direct inspection of $\mathsf{Gen}, \mathsf{Rep}$ and Ver. Below we prove bounded server authentication, user integrity, and privacy separately.

Server Authentication: Suppose for some λ, N there exists a PPT adversary $\mathcal{A}$ with oracle access to $\mathsf{Gen}, \mathsf{Rep}, \mathsf{Gen}^*$ and some polynomial $p(\lambda)$ such that

$$\Pr\left[\mathsf{Rep}(\sigma^*) \neq \perp \wedge \sigma^* \notin \{\sigma_1, \ldots, \sigma_q\} | \sigma^* \leftarrow \mathcal{A}^{\mathsf{Gen}, \mathsf{Rep}, \mathsf{Gen}^*}(N, 1^\lambda)\right] > \frac{1}{p(\lambda)},$$

where $\sigma_1, \ldots, \sigma_q$ are the query response tags from Gen. We show how to use $\mathcal{A}$ to construct $\mathcal{B}$ that forges message authentication codes for $\mathsf{Mac.Tag}$. $\mathcal{B}$ plays the role of the challenger in the server authentication game for $\mathcal{A}$ as follows. Notice that $\mathcal{B}$ has access to the $\mathsf{Mac.Tag}_k$ procedure.

- $\mathcal{B}$ initializes a HELP server by directly sampling $X_{pad} \xleftarrow{\$} \{0,1\}^N$, and setting $\mu = (\mathsf{index} = 1, N)$. It additionally samples a uniform MAC key k^*.
- Whenever $\mathcal{A}$ makes a $\mathsf{Gen}(n)$ query, $\mathcal{B}$ first checks if $\mathsf{index} + n > N$. If so, return $\perp$. Otherwise, $\mathcal{B}$ queries $\mathsf{Mac.Tag}_k$ with (index, n), receiving a tag t. $\mathcal{B}$ sets $y = X_{pad}[\mathsf{index}, \ldots, \mathsf{index} + n - 1]$ and $\sigma = (t, \mathsf{index}, n)$, updates $\mathsf{index} = \mathsf{index} + n$, returns and stores (y, σ).
- Whenever $\mathcal{A}$ makes a $\mathsf{Gen}^*(n)$ query, $\mathcal{B}$ completes the same process as above, but querying $\mathsf{Mac.Tag}_{k^*}$ instead and only returning y.
- Whenever $\mathcal{A}$ makes a $\mathsf{Rep}(\tilde{\sigma})$ query, $\mathcal{B}$ simply checks if there exists an entry (y, σ) in its storage with $\sigma = \tilde{\sigma}$. If so, return $\tilde{y} = y$. Otherwise, return $\tilde{y} = \perp$.
- At the end of the experiment, $\mathcal{A}$ outputs $\sigma^* = (t^*, \mathsf{index}^*, n^*)$. $\mathcal{B}$ simply outputs $m = (\mathsf{index}^*, n^*)$ and $t = t^*$ as the forged tag.

Now we quickly argue that if $\mathcal{A}$ wins the server authentication game, $\mathcal{B}$ wins the MAC forgery game. Notice that $\mathcal{A}$ winning the server authentication game yields that $t^* = \mathsf{Mac.Tag}_k(\mathsf{index}^*, n^*)$ and that $(t^*, \mathsf{index}^*, n^*)$ isn't part of the response of a Gen query. We now have that t^* is a valid tag for (index^*, n^*), so what is left is to show that $\mathcal{B}$ never queried $\mathsf{Mac.Tag}_k$ on (index^*, n^*). Notice that $\mathcal{B}$ queries $\mathsf{Mac.Tag}_k$ only when answering Gen queries. So if $\mathcal{B}$ had queried $\mathsf{Mac.Tag}_k$ on (index^*, n^*) during some Gen query, then it would have responded with $(\mathsf{Mac.Tag}_k(\mathsf{index}^*, n^*), \mathsf{index}^*, n^*) = (t^*, \mathsf{index}^*, n^*)$, which contradicts with the tuple never returned by a Gen query. Therefore, t^* is a valid tag on the tuple (index^*, n^*) that has never been queried before, and hence $\mathcal{B}$ wins the MAC forgery game.

User Integrity: Suppose for some λ there exists a PPT adversarial server $\mathcal{S}$ which wins the user integrity game with advantage $p(\lambda)$ for some polynomial p. We use this to break the collision resistance property of EH as so: To construct $\mathcal{A}$, simply run $\mathcal{S}(1^\lambda)$, receiving $((st, \sigma, y), \tilde{y})$ from $\mathcal{S}$ satisfying $(z, r) \leftarrow \mathsf{Auth}(\sigma, y, 1^\lambda)$ and $\tilde{r} = \mathsf{Ver}(z, \tilde{y})$. Finally, $\mathcal{A}$ outputs $(y, \tilde{y})$.

By construction, we have $(z, r) \leftarrow \mathsf{EH}(y)$ and $(z, \tilde{r}) \leftarrow \mathsf{EH}(\tilde{y})$, but $r \neq \tilde{r}$. We see then that for $y_1 = y$, $y_2 = \tilde{y}$, $(z_1, r_1) = \mathsf{EH}(y_1)$ and $(z_2, r_2) = \mathsf{EH}(y_2)$, we have $z_1 = z = z_2$ yet $r_1 \neq r_2$. So when $\mathcal{S}$ breaks user integrity, $\mathcal{A}$ breaks collision resistance of EH. We conclude that $\mathcal{S}$ must not exist.

Privacy: Privacy comes directly from the extraction property of EH. In the privacy game, notice that the only difference between $b = 0$ and $b = 1$ is whether the adversary receives r_0 or r_1. Therefore, for an adversary to win the privacy game, it needs to distinguish the distribution of (st, σ, z, r_0) from the distribution of (st, σ, z, r_1). Since st and σ are independent from y, z, r_0, and r_1, effectively the adversary needs to distinguish between the distributions of (z, r_0) and (z, r_1). Notice that these two distributions are exactly $(Z, R) = \mathsf{EH}(Y)$ and (Z, U_ℓ). By the extraction property of EH, since $y \sim \{0, 1\}^n$, they are statistically close and hence no adversary can distinguish between them with non-negligible probability.

□

Unbounded Server Authentication. The above construction uses a computational MAC (as in Definition 3) in order to create the randomness tags. This allows the construction to achieve $\tilde{N} = N + \ell_{\mathsf{MAC}}$ at the cost of only satisfying *bounded* server authentication. If instead we use an information-theoretic one-time MAC [GN94] (as in Definition 4), we can achieve *unbounded* server authentication. However, in this case, the length $\tilde{N}$ of randomness needed for N bits of transmitted randomness will be $\tilde{N} = O(N)$, where the constant factor in front of N can be made smaller and smaller, by placing a lower bound on the minimal value $\ell = \Omega(\lambda)$ allowed for $\mathsf{Gen}(\ell)$.

In more detail, the server string X will consist of two parts X_{pad} and X_{mac}, where X_{mac} will contain the one-time MAC keys $(k_1, k_2, \ldots)$ uses for successive calls to Gen. And the dynamic server state μ will also contain the index j of the current Gen query, in addition to values index, n. The j-th call to Gen will use the one-time MAC key k_j to tag the tuple (index, n), and increment j (to ensure each k_j is only used at most once).

To argue that $\tilde{N} = O(N)$, we only need to argue that $|X_{mac}| = O(N) = O(|X_{pad}|)$. To see this, recall that unconditional one-time MACs use a key of size $O(\lambda)$ to authenticate messages of length up to exponential in λ, where λ is the security parameter [GN94]. In our case we only tag a couple of indices (index, n), which certainly has size $O(\lambda)$. Thus, the claim follows if we ensure that the length n of each pad Y is $\Omega(\lambda)$. More generally, we only need the overall length N of all pads (say, T of them) requested by all users to satisfy $N = \Omega(T\lambda)$. This is a reasonable requirement, and likely true for most uses.

Using Dictionaries instead of MACs. Yet another possible trade-off for the main construction is to go one step further, and replace MACs (either a single computational, or many one-time) with "zero-time" time IT-MACs, where the key k is the tag of every message. Namely, one needs k to tag any message, but there is no distinctions between different messages. This presents a viable option if we do not want to use *any cryptography at all*, which could be attractive for low-powered devices or ease of implementation and deployment.

However, this optimization comes at a cost in a different dimension: off-loading static storage of mac keys $k_1, k_2 \ldots$ (or as a single computational MAC key k) from the static server storage X to the dynamic server storage μ. Concretely, the server will use a fresh "zero-time" key k_j for each call to Gen, but then store the mapping from k_j to the authenticated message $V_j = (\mathsf{index}, n)$ in some dynamically growing dictionary D. Moreover, if the server has a fresh source of randomness, it does not need to store all one-time keys k_j in X_{mac}, but can sample them on the spot, only adding the map from k_j to V_j to the dictionary D.

When Bob provides a value k_j in Rep, the server will look for a record $V_j = (\mathsf{index}, n)$ in D. If found, it proceeds as before. Otherwise, it returns $\perp$. Thus, since each k_j is used only once and the messages is remembered in the dictionary, the attacker cannot fool the server from retrieving the wrong index index or message length n. Overall, this variant is extremely simple to implement, but forces the server to store an extra dictionary whose size grows with the number of Gen calls. This trade-off may or may not be preferable in various settings.

As another pragmatic option, the server can even accept the key k_j from the user Alice making the Gen request, although this slightly deviates from our syntax, and puts the burden on users to generate good randomness for "zero-time" MAC key. On a positive, this option gives more flexibility for Alice to distribute MAC keys in advance, so the recipient Bob can make the Rep call even before Alice sent her ciphertext. Additionally, using a *single* master key k^* shared in advance, Alice and Bob can derive the required "zero-time" keys k_j pseudorandomly from k^*, and only transmit the nonces needed to derive k_j from k^* over a *public* (but authenticated) computational channel. Such extra flexibility could be attractive in some scenarios, and might justify the need for the server to store a dynamically growing dictionary mapping k_j to V_j.

5 Composing with Message Transmission

Recall that in Figure 1, we imagined running an HELP instance in composition with a Message Transmission protocol. The single-server HELP construction that we present in Section 4 (and also the distributed HELP construction later in Section 7) satisfy the properties defined in Section 3 (and Section 7), but not quite what we promised in Figure 1. For starters, the HELP instance did not explicitly process messages m to be sent/received, but instead focused on generating proper one-time pads r. Of course, Figure 1 takes care of this by using the simple one-time pad encryption, and also sending Bob the values σ, z needed for the pad reconstruction. More importantly, though, HELP definitions in Section 3 (and later in Section 7) completely ignored the computational leakage in step ③), due to the transmission of values $(m \oplus r, \sigma, z)$ to Bob over (only) computationally secure channel. For example, server authentication Definition 8 in Section 3 allowed the attacker to make Gen^* calls, and obtain no leakage of the corresponding tag value σ (which corresponded to honest users using the scheme). In reality, however, the value σ will be sent over channel ③ in Figure 1, which could leak some computational leakage about σ. Similarly, in the privacy Definition 10 in Section 3 the attacker was not allowed to make any Rep queries, which is allowed in step Ⓐ in Figure 1.

In this section, we fill those modeling gaps, and show how composing a HELP instance with a computationally secure Message Transmission protocol achieves the promised everlasting security guarantees we desire from Figure 1. To achieve this goal, in this section we define stronger variants of user integrity, server authentication, and, most importantly, *everlasting* privacy of our compiler illustrated in Figure 1. These stronger properties will fill the gaps in the single-server definitions from Section 3. Crucially, we will show that the stronger properties are always implied by the seemingly weaker properties from Section 3, provided the computationally secure Message Transmission scheme satisfies the widely accepted notion of Universal Composability (UC) security [Can01]. Concretely, we show that given a (computationally-secure) Message Transmission scheme with Universal Composability (UC) security, we can compose it with any HELP scheme satisfying security properties from Section 3, to obtain a Message Transmission scheme with *everlasting* privacy (and correspondingly stronger forms of user integrity and server authentication).

We start by refreshing the minimal UC security background needed to define the Message Transmission functionality. This allows us to formalize our composition in Figure 1, from any HELP instance and UC-secure Message transmission. Then, for each of the security properties (server authentication, user integrity, and privacy), we present their corresponding stronger definitions for our composed scheme (including *everlasting* privacy as opposed to privacy), and show that HELP security from Section 3 and UC-security of message transmission generically satisfy these new security definitions.

5.1 Universal Composability

The central components of our composition is a HELP instance and a UC-secure message transmission protocol, so we first provide a very brief overview of the UC framework below to remind the reader of the related concepts.

In the Universal Composability (UC) framework [Can01], the goal is to model real-world protocols Π as ideal functionalities $\mathcal{F}$. Towards this end, we consider *real-world processes* and *ideal processes*. A protocol Π is said to UC-realize an ideal functionality $\mathcal{F}$ if the real-world process of running the protocol "emulates" the ideal process for the corresponding ideal functionality. In both processes, we use a PPT Interactive Turing Machine (ITM) to represent the program run by one of the parties. An ITM has 4 different tapes: the input and output tapes model the inputs and outputs that the machine receives or sends to other programs on the same machine, while the incoming and outgoing message tapes model the messages received and sent over the network. Below, we (very briefly) describe the two processes (also illustrated in Figure 2) following the high-level description presented in [CF01]. For a full and formal treatment of the UC framework, we refer the readers to Canetti's original paper [Can01].

Real-World Process. In both the real-world and ideal processes, we consider a computational environment $\mathcal{E}$, which can be thought of as a PPT distinguisher trying to distinguish between the real-world and the ideal process. In the real-world process, we consider some honest parties $\mathcal{P}_1, \mathcal{P}_2, \ldots$ executing the protocol Π with some adversary $\mathcal{A}$ and the environment $\mathcal{E}$. All of the participants are in PPT of some security parameter λ.

The execution of the process proceeds through a sequence of *activations*, where one participant ($\mathcal{A}, \mathcal{E}$, or one of the $\mathcal{P}_i$'s) is activated each time. The activated participant may read from its own input and incoming message tape, execute its code, and then possibly write on its own output or outgoing message tape. The environment $\mathcal{E}$ and the adversary $\mathcal{A}$ have additional capabilities. Additionally, the environment $\mathcal{E}$ can write on the input tapes of the honest parties or the adversary and read from their output tapes. The adversary $\mathcal{A}$ can read messages from the outgoing message tapes of the honest parties and copy them to the incoming message tapes of the recipient party. Notice that the adversary $\mathcal{A}$ is not allowed to modify or duplicate the messages – only the original messages produced by the honest parties are allowed. The adversary may also corrupt honest parties, gaining full control of the party, as well as its internal information.

In the execution of the process, the environment $\mathcal{E}$ is activated first with the input x on its input tape. Once activated, the environment may write onto the input tape of either one of the honest parties or the adversary. Then, that participant is activated next. If no input tape is written onto, then the execution halts. Every time an honest party finishes activation, the environment is activated automatically. When the adversary delivers a message to some honest party $\mathcal{P}_i$ during activation, $\mathcal{P}_i$ will be activated next. If

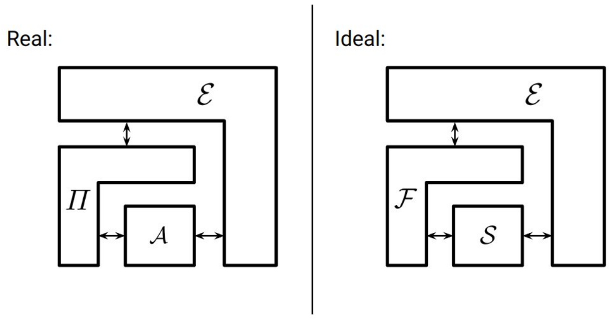


Figure 2: Illustration of the real-world process vs. the ideal process in the UC framework.

the adversary does not write onto any incoming message tape during activation, then the environment is activated right after. Notice this allows the environment and the adversary to freely exchange information using the adversary's input and output tape. The output of the experiment is a single bit output by $\mathcal{E}$, which we denote as $\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{E}}(\lambda, x)$.

Ideal Process. We now describe the ideal process, which centers around an ideal functionality $\mathcal{F}$ that captures the desired functionality. This $\mathcal{F}$ is modeled as another ITM that interacts with the environment $\mathcal{E}$, an *ideal process adversary* $\mathcal{S}$, and a set of *dummy parties* $\tilde{\mathcal{P}}_1, \tilde{\mathcal{P}}_2, \ldots$. The dummy parties have a very simple fixed behavior: upon activation with an input, it simply forwards the input to $\mathcal{F}$ by copying the input to its own outgoing message tape; upon activation with an incoming message from $\mathcal{F}$, it simply copies the message to its own output tape. $\mathcal{F}$ receives the messages from the dummy parties by directly reading from their outgoing message tapes and sends messages to them by directly writing on their incoming message tapes. The ideal process adversary $\mathcal{S}$ behaves similarly to a real-world adversary $\mathcal{A}$, except that it cannot read the incoming and outgoing message tapes of $\mathcal{F}$ and the dummy parties. Here instead, $\mathcal{F}$ is in charge of delivering the messages between $\mathcal{F}$ and the dummy parties. As in the real-world process, $\mathcal{S}$ can also corrupt dummy parties.

The execution of the idea process is similar to that of a real-world process. Notice that there is no direct communication between the dummy parties – all communications are achieved through the ideal functionality $\mathcal{F}$. The output of the experiment is also a bit by $\mathcal{E}$, which we denote as $\mathsf{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{E}}(\lambda, x)$.

Definition 14 (Universal Composability [Can01])**.** Let $\mathcal{F}$ be an ideal functionality, and let Π be an implementation. We say that Π γ-UC-realizes $\mathcal{F}$ if, for any security parameter λ and any real-world adversary $\mathcal{A}$ there exists an ideal-process adversary $\mathcal{S}$ such that for any environment $\mathcal{E}$ and any input x we have

$$|\Pr[\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{E}}(\lambda, x) = 1] - \Pr[\mathsf{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{E}}(\lambda, x) = 1]| \leq \gamma.$$

The above definition can be simplified with a *dummy adversary* $\mathcal{A}_{\mathsf{dummy}}$. A dummy adversary's behavior is very simple: it just forwards messages from the environment to the designated parties, and also from the parties back to the environment. With this dummy adversary, the environment has almost full control of the protocol, and hence simulating this dummy adversary will be the hardest, therefore capturing the "for all" quantifier on the real-world adversaries.

Definition 15 (Universal Composability (UC) with $\mathcal{A}_{\mathsf{dummy}}$)**.** Let $\mathcal{F}$ be an ideal functionality and let Π be an implementation. We say Π γ-UC-realizes $\mathcal{F}$ if for any security parameter λ, there exists an ideal-process adversary $\mathcal{S}$ such that for any environment $\mathcal{E}$

and any input x we have

$$\left|\Pr[\mathsf{Real}_{\Pi,\mathcal{A}_{\mathsf{dummy}},\mathcal{E}}(\lambda,x)=1]-\Pr[\mathsf{Ideal}_{\mathcal{F},\mathcal{S},\mathcal{E}}(\lambda,x)=1]\right|\le\gamma.$$

5.2 A Compiler for Secure Message Transmission

Now we show how to augment a UC-secure Message Transmission protocol Π to obtain a message transmission Π' with everlasting privacy using a HELP.

We first define the ideal functionality for secure message transmission $\mathcal{F}_{\mathsf{MT}}$.

Functionality $\mathcal{F}_{\mathsf{MT}}$

$\mathcal{F}_{\mathsf{MT}}$ proceeds as follows, running with parties $\mathcal{P}_1,\dots,\mathcal{P}_n$ and an adversary $\mathcal{S}$.

1. Upon receiving a tuple $(\mathtt{Send},m,\mathcal{P}_j)$ from $\mathcal{P}_i$, leak $(\mathcal{P}_i,\mathcal{P}_j,|m|)$ to $\mathcal{S}$.
2. When $\mathcal{S}$ returns $\mathtt{OK}$, output $(\mathtt{Sent},m)$ to $\mathcal{P}_j$.

We next show how to construct a message transmission protocol Π' with everlasting privacy, by using a message transmission protocol Π that UC-realizes $\mathcal{F}_{\mathsf{MT}}$ and a HELP instance HELP as ingredients.

Construction 5.1. *Let λ be the security parameter. Let Π be a message transmission protocol, and HELP be a HELP instance. We construct message transmission protocol Π' as follows:*

- ***To send a message** m **to** $\mathcal{P}_j$:*
  1. *Compute $n\leftarrow\mathsf{PadLength}(|m|,1^\lambda)$;*
  2. *Send a $\mathsf{Gen}(n)$ query to HELP, and receive (σ,y);*
  3. *Compute $(z,r)\leftarrow\mathsf{Auth}(\sigma,y,1^\lambda)$;*
  4. *Run protocol Π to send the tuple $(m\oplus r,\sigma,z)$ to $\mathcal{P}_j$.*
- ***Upon receiving** $\tilde{m}$ **from** Π:*
  1. *Parse $\tilde{m}=(x,\tilde{\sigma},\tilde{z})$;*
  2. *Send a $\mathsf{Rep}(\tilde{\sigma})$ query to HELP, and receive $\tilde{y}$;*
  3. *Compute $\tilde{r}\leftarrow\mathsf{Ver}(\tilde{z},\tilde{y})$;*
  4. *If $\tilde{r}=\perp$, abort and output $\perp$;*
  5. *Output $m'=x\oplus\tilde{r}$.*

Correctness requires that $m'=m$ and follows trivially from the correctness of Π and HELP. We discuss the three desired security properties for this construction in the following three subsections.

5.3 Server Authentication

5.3.1 Definition

First, let us recall the original definition of server authentication.

Definition 8. *Let λ be the security parameter. We say that HELP satisfies {bounded, unbounded} ε*-server authentication *for $\varepsilon=\mathtt{negl}(\lambda)$ if, for all {PPT, unbounded} $\mathcal{A}$ with oracle access to $\mathsf{Gen},\mathsf{Rep},\mathsf{Gen}^*$, and any value of N, after we run $\mathsf{Init}(N,1^\lambda)$ to initialize (X,μ), we have*

$$\Pr\left[\ \mathsf{Rep}(\sigma^*)\neq\perp\wedge\sigma^*\notin\{\sigma_1,\dots,\sigma_q\}\ \middle|\ \sigma^*\leftarrow\mathcal{A}^{\mathsf{Gen},\mathsf{Rep},\mathsf{Gen}^*}(N,1^\lambda)\ \right]\le\varepsilon,$$

where $\mathsf{Gen}, \mathsf{Rep}$ *are defined as before,* $\mathsf{Gen}^*(n)$ *calls* $(\sigma, y) \leftarrow \mathsf{Gen}(n)$ *but only returns* y*, and* $\{\sigma_1, \ldots, \sigma_q\}$ *denotes the set of tags returned in response to the adversary's queries to* Gen*.*

Notice that this definition has a slight mismatch from Figure 1. In the definition above, the adversary with access to the Gen^* oracles (recall that these are used to model queries made to Gen by honest users) can only get the y values, not the σ's. However, in Figure 1 and Construction 5.1, when the honest users make Gen queries, σ is sent through the message transmission protocol. In this case, the adversary *does* receive computational leakage of σ from the message transmission protocol, which is not captured by the definition above.

With that in mind, we modify the server authentication definition as follows to fit into the composed protocol.

Definition 16. Let λ be the security parameter. We say a HELP-aided message transmission protocol MT has ε*-server authentication* for $\varepsilon = \mathtt{negl}(\lambda)$ if for all PPT adversaries $\mathcal{A}$ and choices of N (the parameter in HELP), we have

$$Pr\left[\mathsf{ExptSvrAuth}_{\mathcal{C},\mathcal{A}}^{\mathsf{HELP},\mathsf{MT}}(N, \lambda) = 1\right] \leq \varepsilon,$$

where $\mathsf{ExptSvrAuth}$ is specified in Figure 3.

The key difference is in Step 2c, the challenger runs MT to send m_i, which could potentially leak information about the HELP tag σ used by the challenger.

5.3.2 Proving Server Authentication

We first prove that our construction satisfies server authentication after the composition.

Theorem 3. *If* HELP *has* ε*-server authentication, and* Π γ*-UC-realizes* $\mathcal{F}_{\mathsf{MT}}$*, then the* HELP*-aided message transmission protocol* Π' *in Construction 5.1 has* $(\varepsilon + \gamma)$*-server authentication.*

Proof. The intuition behind the proof is that we want to reduce this to the server authentication of HELP, but notice that in the $\mathsf{ExptSvrAuth}$ experiment, $\mathcal{A}$ can potentially get some additional information from step 2c than in the original server authentication definition (Definition 8. Therefore, we take one extra step by invoking the UC-security of Π and replacing the leakage with just the length. Formally, we prove this through the following hybrids.

- H_1: The same as $\mathsf{ExptSvrAuth}_{\mathcal{C},\mathcal{A}}^{\mathsf{HELP},\Pi'}$.
- H_2: Now we replace the message transmission protocol Π with the ideal functionality $\mathcal{F}_{\mathsf{MT}}$. Specifically, we change step 2(c)iv to the following:

  2(c)iv. Send the tuple $(\mathtt{Send}, (m_i \oplus r_i^*, \sigma_i^*, z_i^*), \cdot)$ to $\mathcal{F}_{\mathsf{MT}}$.

In the rest of the proof, we first bound the probability that any PPT adversary $\mathcal{A}$ can distinguish between H_1 and H_2, and then bound the probability of the adversary winning the game in H_2, denoted as $\Pr_{H_2}[\mathsf{BREAK}]$.

First, we show that if a PPT adversary $\mathcal{A}$ can distinguish between H_1 and H_2 with probability γ', then for all ideal-process adversary $\mathcal{S}$, there exists an environment machine $\mathcal{E}$ that can distinguish between $(\mathcal{F}, \mathcal{S})$ and $(\Pi, \mathcal{A}_{\mathsf{dummy}})$ with probability $\gamma = \gamma'$. The high-level idea is that the environment machine $\mathcal{E}$ will simulate either H_1 or H_2 for $\mathcal{A}$ and construct its own output based on the output of $\mathcal{A}$. Specifically, given $\mathcal{A}$, we construct $\mathcal{E}$ as follows by simulating the view for $\mathcal{A}$. Most of the steps are directly simulating $\mathsf{ExptSvrAuth}$, so we highlight the main differences in red.

$\mathsf{ExptSvrAuth}^{\mathsf{HELP},\mathsf{MT}}_{\mathcal{C},\mathcal{A}}(N,\lambda)$:

1. The challenger $\mathcal{C}$ runs $(X,\mu) \leftarrow \mathsf{Init}(N, 1^\lambda)$.
2. For a polynomial number of rounds $i = 1, 2, \ldots, q$, the adversary $\mathcal{A}$ may choose one of the following:
   (a) **Submit a $\mathsf{Gen}(n_i)$ query to $\mathcal{C}$:**
   $\mathcal{C}$ computes $(\sigma_i, y_i) \leftarrow \mathsf{Gen}\,(n_i)$, and responds with (σ_i, y_i).
   (b) **Submit a $\mathsf{Rep}(\tilde{\sigma}_i)$ query to $\mathcal{C}$:**
   $\mathcal{C}$ computes $\tilde{y}_i \leftarrow \mathsf{Rep}\,(\tilde{\sigma}_i)$, and responds with $\tilde{y}_i$.
   (c) **Submit a message m_i to $\mathcal{C}$:**
   $\mathcal{C}$ runs MT to send m_i, and $\mathcal{A}$ receives the corresponding leakage from MT. Specifically, $\mathcal{C}$ computes the following:
      i. $n_i^* \leftarrow \mathsf{PadLength}(|m_i|, 1^\lambda)$;
      ii. $(\sigma_i^*, y_i^*) \leftarrow \mathsf{Gen}(n_i^*)$;
      iii. $(z_i^*, r_i^*) \leftarrow \mathsf{Auth}(\sigma_i^*, y_i^*, 1^\lambda)$;
      iv. Run the underlying message transmission protocol Π to send the tuple $(m_i \oplus r_i^*, \sigma_i^*, z_i^*)$. [a][b]
3. $\mathcal{A}$ wins the game (the experiment outputs 1) if and only if $\mathcal{A}$ submits some $\mathsf{Rep}(\tilde{\sigma}_i)$ query and receives a non-$\perp$ response, but $\tilde{\sigma}_i$ is not a tag returned by a previous Gen query, i.e. $\mathcal{A}$ forges the tag $\tilde{\sigma}_i$. Put formally, $\mathcal{A}$ wins the game if and only if
$$\exists i \in [q] \text{ s.t. } \tilde{\sigma}_i \notin \{\sigma_j\}_{j\in[i]} \wedge \tilde{y}_i \neq \perp.$$

[a] Typically, $\mathcal{C}$ would also need to specify a party $\mathcal{P}_j$ to receive the tuple. But here we omit the party, as we only care about the leakage caused by running the protocol. If it helps, a reader could think about here $\mathcal{C}$ runs Π to send the tuple to itself.

[b] In the case where Π is interactive, the adversary $\mathcal{A}$ is also allowed to make other Gen and Rep queries asynchronously during the execution of Π.

Figure 3: Security experiment $\mathsf{ExptSvrAuth}$.

1. Run $(X,\mu) \leftarrow \mathsf{Init}(N, 1^\lambda)$.
2. To answer $\mathcal{A}$'s queries:
   (a) Whenever $\mathcal{A}$ submits a $\mathsf{Gen}(n_i)$ query, compute $(\sigma_i, y_i) \leftarrow \mathsf{Gen}(n_i)$, and respond with (σ_i, y_i).
   (b) Whenever $\mathcal{A}$ submits a $\mathsf{Rep}(\tilde{\sigma}_i)$ query, compute $\tilde{y}_i \leftarrow \mathsf{Rep}\,(\tilde{\sigma}_i)$, and respond with $\tilde{y}_i$.
   (c) Whenever $\mathcal{A}$ submits a message query m_i, compute $n_i^* \leftarrow \mathsf{PadLength}(|m_i|, 1^\lambda)$, $(\sigma_i^*, y_i^*) \leftarrow \mathsf{Gen}(n_i^*)$, $(z_i^*, r_i^*) \leftarrow \mathsf{Auth}(\sigma_i^*, y_i^*, 1^\lambda)$. Invoke either $\mathcal{A}_{\mathsf{dummy}}$ or $\mathcal{S}$ to have an honest party send $(m_i \oplus r_i^*, \sigma_i^*, z_i^*)$, $\mathcal{A}_{\mathsf{dummy}}$ or $\mathcal{S}$ will receive a leakage τ_i which is also forwarded back to the environment. Send τ_i to $\mathcal{A}$.
3. At the end of the experiment, if $\mathcal{A}$ outputs it is in H_1, output that $\mathcal{E}$ is in the real-world process interacting with $\mathcal{A}_{\mathsf{dummy}}$. Otherwise, output that it is in the ideal process interacting with $\mathcal{S}$.

Notice that if $\mathcal{E}$ were in the ideal process, then $\tau_i = |(m_i \oplus r_i^*, \sigma_i^*, z_i^*)|$, which matches what $\mathcal{A}$ expects from step 2c. On the other hand, if $\mathcal{E}$ were in the real-world process, then τ_i is whatever is leaked to the adversary through the execution of Π, which also matched what $\mathcal{A}$ expects. Therefore, the view is simulated correctly for $\mathcal{A}$. If $\mathcal{A}$ successfully distinguishes between H_1 and H_2, $\mathcal{E}$ also successfully distinguishes between real and ideal. Since for PPT $\mathcal{E}$, the real process and the ideal process are γ-close, H_1 and H_2 are also γ close for PPT $\mathcal{A}$.

Next we finish the proof by showing that in H_2 the probability of the adversary winning is just ε. This is by direct reduction to the server authentication of HELP. Specifically, we show that an adversary $\mathcal{A}$ that wins H_2 implies an adversary $\mathcal{A}'$ that breaks the server authentication of HELP. $\mathcal{A}'$ simulates the view for $\mathcal{A}$ by simply forwarding all the Gen and Rep queries made by $\mathcal{A}$ to the oracles that $\mathcal{A}'$ has access to and correspondingly forwarding the oracle responses. Upon receiving m_i from $\mathcal{A}$, simply leak $(\mathtt{Sent}, |m_i|)$ to $\mathcal{A}$. By the end of the experiment, $\mathcal{A}$ has won the game by querying $\tilde{\sigma}$ that is not returned by a previous Gen query and has a non-$\perp$ response. $\mathcal{A}'$ simply outputs $\tilde{\sigma}$ and wins the game. Therefore, the probability of $\mathcal{A}$ breaking H_2 is at most the probability that $\mathcal{A}'$ can break server authentication, i.e. $\Pr_{H_2}[\mathsf{BREAK}] \leq \varepsilon$.

Combining the previous two parts, we have $\Pr\left[\mathsf{ExptSvrAuth}_{\mathcal{C},\mathcal{A}}^{\mathsf{HELP},\mathsf{MT}}(N,\lambda)=1\right] \leq \gamma+\varepsilon$ as desired. □

5.4 User Integrity

5.4.1 Definition

Similar to server authentication, the notion of user integrity would also need to be adjusted accordingly for the composed setting. On a high level, the original user integrity definition dictates that a malicious server cannot fool the user into producing a different yet valid random key r. In the composed setting, we would like to modify the definition to capture the user integrity for *messages*, as opposed to keys. Namely, we want that an adversarial HELP server cannot cause a transmitted message to be received as a different one. Specifically, we define it as below.

Definition 17. Let λ be the security parameter. We say that HELP satisfies δ-*user integrity* with $\delta = \mathtt{negl}(\lambda)$, if for all PPT servers $\mathcal{S}$ and choices of N, we have

$$\Pr\left[\mathsf{ExptUsrInt}_{\mathcal{C},\mathcal{S}}^{\mathsf{HELP},\mathsf{MT}}(N,\lambda)=1\right] \leq \delta,$$

where $\mathsf{ExptUsrInt}$ is specified in Figure 4.

5.4.2 Proving User Integrity

We prove user integrity of our Construction 5.1 through the following theorem.

Theorem 4. *If* HELP *has* δ*-server authentication, and* Π γ*-UC-realizes* $\mathcal{F}_{\mathsf{MT}}$*, then the* HELP*-aided message transmission protocol* Π' *in Construction 5.1 has* $(\delta+\gamma)$*-user integrity.*

Proof. The first step, similar to the proof of Server Authentication for Theorem 3, is to invoke the UC-security of the underlying message transmission protocol Π to replace it with the ideal functionality $\mathcal{F}_{\mathsf{MT}}$. This step introduces an error of γ.

Once we have the ideal functionality $\mathcal{F}_{\mathsf{MT}}$, it is guaranteed that the tuple $(m \oplus r, \sigma, z)$ is received as is, and therefore we have $x = m \oplus r, \tilde{\sigma} = \sigma$, and $\tilde{z} = z$[8].

Then, we can reduce to the user integrity of the HELP instance. Specifically, if an adversary $\mathcal{S}$ is able to win the composed user integrity game with ideal message transmission

[8] Notice that in fact, only integrity of $m \oplus r$ and z are necessary for the proof. So practically, σ can be sent through some message transmission protocol with no integrity guarantees (secrecy is still needed for the server authentication property though).

$\mathsf{ExptUsrInt}_{\mathcal{C},\mathcal{S}}^{\mathsf{HELP},\mathsf{MT}}(N, \lambda)$:

1. The adversary $\mathcal{S}$ is initialized with the security parameter 1^λ.
2. The challenger $\mathcal{C}$ samples message m, and computes $n \leftarrow \mathsf{PadLength}(|m|, 1^\lambda)$.
3. The challenger $\mathcal{C}$ submits the $\mathsf{Gen}(n)$ query.
4. The adversary $\mathcal{S}$ on input n produces the response (σ, y), which the challenger $\mathcal{C}$ receives.
5. The challenger computes $(z, r) \leftarrow \mathsf{Auth}(\sigma, y, 1^\lambda)$, and runs the underlying message transmission protocol Π to send $(m \oplus r, \sigma, z)$, which will be received as $(x, \tilde{\sigma}, \tilde{z})$.
6. The challenger $\mathcal{C}$ submits the $\mathsf{Rep}(\tilde{\sigma})$ query.
7. The adversary $\mathcal{S}$ on input $\tilde{\sigma}$ produces the response $\tilde{y}$, which the challenger $\mathcal{C}$ receives.
8. The challenger $\mathcal{C}$ computes $\tilde{r} \leftarrow \mathsf{Ver}(\tilde{z}, \tilde{y})$. The adversary wins the game (and the experiment outputs 1) if and only if $\tilde{r} \neq \perp$ and $x \oplus \tilde{r} \neq m$.

Figure 4: Security experiment $\mathsf{ExptUsrInt}$.

functionality, then we can build an adversary $\mathcal{S}'$ that wins the original user integrity game. $\mathcal{S}'$ will use $\mathcal{S}$ as a subroutine by playing the role of the challenger in the $\mathsf{ExptUsrInt}$ game as follow:

1. $\mathcal{S}'$ receives the security parameter 1^λ, which it uses to initialize $\mathcal{S}$.
2. $\mathcal{S}'$ samples message m, computes $n \leftarrow \mathsf{PadLength}(|m|, 1^\lambda)$, and submits the $\mathsf{Gen}(n)$ query to $\mathcal{S}$.
3. The adversary $\mathcal{S}$ responds with (σ, y), which $\mathcal{S}'$ also outputs. Additionally, $\mathcal{S}'$ outputs a state $st = \sigma$.
4. $\mathcal{S}'$ receives z, r together with $st = \sigma$. It submits the $\mathsf{Rep}(\sigma)$ query to $\mathcal{S}$.
5. The adversary $\mathcal{S}$ responds with $\tilde{y}$, which $\mathcal{S}'$ also outputs.

Now we briefly argue that if $\mathcal{S}$ wins the $\mathsf{ExptUsrInt}$ game, then $\mathcal{S}'$ wins the original user integrity game. $\mathcal{S}$ wins the game only if $\tilde{r} \neq \perp$ and $\tilde{r} \neq m \oplus x = r$ for $\tilde{r} \leftarrow \mathsf{Ver}(z, \tilde{y})$. This immediately gives $\tilde{r} \notin \{r, \perp\}$ as desired.

Therefore, bringing the two parts together, Construction 5.1 has $(\delta + \gamma)$-user integrity. □

5.5 Everlasting Privacy

5.5.1 Definition

The overall idea behind everlasting privacy is to capture the security model illustrated in Figure 1 using an indistinguishability-based game definition. We define the security game as a two-stage experiment. First, the adversary $\mathcal{A}_1$ is PPT but has access to the HELP instance. To capture $\mathcal{A}_1$'s ability to communicate with the HELP instance arbitrarily, we allow $\mathcal{A}_1$ arbitrary adaptive Gen and Rep queries, before and after the challenge. In the first stage, $\mathcal{A}_1$ also chooses two challenge messages of equal length. The challenger picks a random one to send using the message transmission protocol. At the end of the first stage, $\mathcal{A}_1$ outputs its own view, which is then passed to the second-stage adversary $\mathcal{A}_2$. The adversary $\mathcal{A}_2$ is computationally unbounded but no longer has access to the HELP instance. The goal is for $\mathcal{A}_2$ to successfully guess which of the two challenge messages were sent in the first stage by solely depending on the view of $\mathcal{A}_1$ in the first stage. Put formally, we define everlasting privacy as follows.

Definition 18. Let λ be the security parameter. We say a HELP-aided message transmission protocol MT has *ω-everlasting privacy* for $\omega = \mathtt{negl}(\lambda)$ if for all adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ with PPT $\mathcal{A}_1$ and unbounded $\mathcal{A}_2$, and choices of N (the parameter in HELP), we have

$$\left|\Pr[\mathsf{ExptEvltPriv}^{\mathsf{HELP},\mathsf{MT}}_{\mathcal{C},\mathcal{A},1}(N,\lambda) = 1] - \Pr[\mathsf{ExptEvltPriv}^{\mathsf{HELP},\mathsf{MT}}_{\mathcal{C},\mathcal{A},0}(N,\lambda) = 1]\right| \le \omega,$$

where $\mathsf{ExptEvltPriv}$ is specified in Figure 5.

5.5.2 Proving Everlasting Security

Now we formally prove that our construction of Π' in Construction 5.1 has everlasting privacy.

Theorem 5. *If* HELP *has ε-server authentication and ξ-privacy, and* Π *γ-UC-realizes* $\mathcal{F}_{\mathsf{MT}}$*, then the* HELP*-aided message transmission protocol* Π' *in Construction 5.1 has* $(2\varepsilon + 2\gamma + 4\xi)$*-everlasting privacy.*

Proof. We structure the proof around an invocation of Lemma 1.

We first define a $\mathsf{BREAK}(\mathsf{ExptEvltPriv}^{\mathsf{HELP},\Pi'}_{\mathcal{C},\mathcal{A},b})$ predicate to indicate whether $\mathcal{A}_1$ has broken server authentication in experiment $\mathsf{ExptEvltPriv}^{\mathsf{HELP},\Pi'}_{\mathcal{C},\mathcal{A},b}$. For simplicity, we shorthand it as just BREAK_b. Specifically, we have

$$\mathsf{BREAK}_b := \exists\ i \in [q_2].\ \big(\tilde{\sigma}_i \notin \{\sigma_j\}_{j\in[i]} \wedge \tilde{y}_i \neq \bot\big)\,.$$

Notice that $\Pr[\mathsf{BREAK}_0]$ and $\Pr[\mathsf{BREAK}_1]$ are both bounded by the server authentication error. And by Theorem 3, since HELP has ε-server authentication and Π γ-UC-realizes $\mathcal{F}_{\mathsf{MT}}$, Π' has $(\varepsilon + \gamma)$-server authentication. Hence, we have $\Pr[\mathsf{BREAK}_0] = \Pr[\mathsf{BREAK}_1] \le \varepsilon + \gamma$.

Invoking Lemma 1, let A_b be the event that $\mathsf{ExptEvltPriv}^{\mathsf{HELP},\Pi'}_{\mathcal{C},\mathcal{A},b}(N,\lambda) = 1$, and B_b be simply $\neg\mathsf{BREAK}_b$. Then by Lemma 1,

$$\begin{aligned}&\left|\Pr[\mathsf{ExptEvltPriv}^{\mathsf{HELP},\Pi'}_{\mathcal{C},\mathcal{A},1}(N,\lambda) = 1] - \Pr[\mathsf{ExptEvltPriv}^{\mathsf{HELP},\Pi'}_{\mathcal{C},\mathcal{A},0}(N,\lambda) = 1]\right|\\ &\le \Pr[\mathsf{BREAK}_0] + \Pr[\mathsf{BREAK}_1] + |\Pr[A_0 \wedge \neg\mathsf{BREAK}_0] - \Pr[A_1 \wedge \neg\mathsf{BREAK}_1]|\end{aligned}$$

Since we already have $\Pr[\mathsf{BREAK}_0], \Pr[\mathsf{BREAK}_1] \le \varepsilon + \gamma$ from server authentication, we just need to show that $|\Pr[A_0 \wedge \neg\mathsf{BREAK}_0] - \Pr[A_1 \wedge \neg\mathsf{BREAK}_1]|$ is negligible through the following Lemma.

Lemma 2. *If* HELP *has ξ-privacy, then*

$$|\Pr[A_0 \wedge \neg\mathsf{BREAK}_0] - \Pr[A_1 \wedge \neg\mathsf{BREAK}_1]| \le 4\xi.$$

On a high level, we want to reduce this to the privacy of the underlying HELP scheme. But the challenge is that in the everlasting privacy game the adversary $\mathcal{A}_1$ is allowed Rep queries, while the adversary $\mathcal{A}$ in the privacy game is not. We handle this through a sequence of hybrid. In all of the hybrids, we have the outputs of the experiments set

$\mathsf{ExptEvltPriv}^{\mathsf{HELP},\mathsf{MT}}_{\mathcal{C},\mathcal{A}=(\mathcal{A}_1,\mathcal{A}_2),b}(N,\lambda)$:

1. The challenger $\mathcal{C}$ runs $(X,\mu) \leftarrow \mathsf{Init}(N, 1^\lambda)$.
2. For a polynomial number of rounds $i = 1, 2, \ldots, q_1$, the adversary $\mathcal{A}_1$ may choose one of the following:
   (a) **Submit a $\mathsf{Gen}(n_i)$ query to $\mathcal{C}$:**
   $\mathcal{C}$ computes $(\sigma_i, y_i) \leftarrow \mathsf{Gen}\,(n_i)$, and responds with (σ_i, y_i).
   (b) **Submit a $\mathsf{Rep}(\tilde{\sigma}_i)$ query to $\mathcal{C}$:**
   $\mathcal{C}$ computes $\tilde{y}_i \leftarrow \mathsf{Rep}\,(\tilde{\sigma}_i)$, and responds with $\tilde{y}_i$.
3. $\mathcal{A}_1$ chooses two messages m_0, m_1 with $|m_0| = |m_1|$ and sends m_0, m_1 to $\mathcal{C}$.
4. The challenger $\mathcal{C}$ runs MT to send m_b, and $\mathcal{A}_1$ receives the corresponding leakage from MT.
   Specifically, $\mathcal{C}$ computes the following:
   (a) $n \leftarrow \mathsf{PadLength}(|m_b|, 1^\lambda)$;
   (b) $(\sigma, y) \leftarrow \mathsf{Gen}(n^*)$;
   (c) $(z, r) \leftarrow \mathsf{Auth}(\sigma, y, 1^\lambda)$;
   (d) Run the underlying message transmission protocol Π to send the tuple $(m_b \oplus r, \sigma, z)$. [a]
5. Again, for a polynomial number of rounds $i = q_1 + 1, q_1 + 2, \ldots, q_2$, the adversary $\mathcal{A}_1$ may choose one of the following:
   (a) **Submit a $\mathsf{Gen}(n_i)$ query to $\mathcal{C}$:**
   $\mathcal{C}$ computes $(\sigma_i, y_i) \leftarrow \mathsf{Gen}\,(n_i)$, and responds with (σ_i, y_i).
   (b) **Submit a $\mathsf{Rep}(\tilde{\sigma}_i)$ query to $\mathcal{C}$:**
   $\mathcal{C}$ computes $\tilde{y}_i \leftarrow \mathsf{Rep}\,(\tilde{\sigma}_i)$, and responds with $\tilde{y}_i$.
6. $\mathcal{A}_2$ takes as input $\mathsf{view}(\mathcal{A}_1)$ and outputs a bit, the output of the experiment.

[a] Similar to server authentication, in the case where Π is interactive, the adversary $\mathcal{A}$ is also allowed to make other Gen and Rep queries asynchronously during the execution of Π.

Figure 5: Security experiment $\mathsf{ExptEvltPriv}$.

to be the logical AND of the adversary's output and the BREAK predicate. First of all, we modify the experiment so that Rep queries simply return $\perp$ on all $\tilde{\sigma}$'s that are not the result of previous Gen queries, but when we calculate the BREAK predicate, we still calculate it based on the old query responses. We argue that this change results in identical output distributions of the experiments. Then, we can reduce to the privacy game, as now the Rep queries do not provide any useful information, and thus can be easily simulated by the adversary in the privacy game.

Proof. We prove this through a sequence of hybrids. As an overview, in H_1, we have $\mathsf{ExptEvltPriv}^{\mathsf{HELP},\Pi'}_{\mathcal{C},\mathcal{A},0}$ except that we adjust the output to reflect the $\neg\mathsf{BREAK}_0$ predicate. In H_2, we reply $\perp$ to the Rep queries that would lead to BREAK_0. In H_3, we switch from sending m_0 to random. In H_4, we switch from random back to m_1. In H_5, we revert the changes made in H_2 and get back $\mathsf{ExptEvltPriv}^{\mathsf{HELP},\Pi'}_{\mathcal{C},\mathcal{A},1}$ with the $\neg\mathsf{BREAK}_1$ adjusted into the output. We will show that the outputs of H_1 and H_2, H_4 and H_5 are identically distributed, while the outputs of H_2 and H_3, H_3 and H_4 are statistically close. We detail the hybrids below:

- H_1: The same as $\mathsf{ExptEvltPriv}^{\mathsf{HELP},\Pi'}_{\mathcal{C},\mathcal{A},0}$, except that the output of experiment is adjusted to reflect $\mathsf{ExptEvltPriv}^{\mathsf{HELP},\Pi'}_{\mathcal{C},\mathcal{A},0} \wedge \neg\mathsf{BREAK}_0$. Concretely, it is as below, with the adjustment highlighted in red:

  H_1:

  1. The challenger $\mathcal{C}$ runs $(X, \mu) \leftarrow \mathsf{Init}(N, 1^\lambda)$.
  2. For a polynomial number of rounds $i = 1, 2, \ldots, q_1$, the adversary $\mathcal{A}_1$ may choose one of the following:
     (a) **Submit a $\mathsf{Gen}(n_i)$ query to $\mathcal{C}$:**
     $\mathcal{C}$ computes $(\sigma_i, y_i) \leftarrow \mathsf{Gen}\,(n_i)$, and responds with (σ_i, y_i).
     (b) **Submit a $\mathsf{Rep}(\tilde{\sigma}_i)$ query to $\mathcal{C}$:**
     $\mathcal{C}$ computes $\tilde{y}_i \leftarrow \mathsf{Rep}\,(\tilde{\sigma}_i)$, and responds with $\tilde{y}_i$.
  3. The adversary $\mathcal{A}_1$ chooses two messages m_0, m_1 with $|m_0| = |m_1|$ and sends m_0, m_1 to $\mathcal{C}$.
  4. The challenger $\mathcal{C}$ runs the following:
     (a) $n \leftarrow \mathsf{PadLength}(|m_0|, 1^\lambda)$;
     (b) $(\sigma, y) \leftarrow \mathsf{Gen}(n)$;
     (c) $(z, r) \leftarrow \mathsf{Auth}(\sigma, y, 1^\lambda)$;
     (d) Run the message transmission protocol Π to send the tuple $(m_0 \oplus r, \sigma, z)$.
  5. Again, for a polynomial number of rounds $i = q_1 + 1, q_1 + 2, \ldots, q_2$, the adversary $\mathcal{A}_1$ may choose one of the following:
     (a) **Submit a $\mathsf{Gen}(n_i)$ query to $\mathcal{C}$:**
     $\mathcal{C}$ computes $(\sigma_i, y_i) \leftarrow \mathsf{Gen}\,(n_i)$, and responds with (σ_i, y_i).
     (b) **Submit a $\mathsf{Rep}(\tilde{\sigma}_i)$ query to $\mathcal{C}$:**
     $\mathcal{C}$ computes $\tilde{y}_i \leftarrow \mathsf{Rep}\,(\tilde{\sigma}_i)$, and responds with $\tilde{y}_i$.
  6. The adversary $\mathcal{A}_2$ takes as input $\mathsf{view}(\mathcal{A}_1)$ and sends a bit b' to $\mathcal{C}$. $\mathcal{C}$ computes $\mathsf{BREAK} = \exists\ i \in [q_2].\,(\tilde{\sigma}_i \notin \{\sigma_j\}_{j\in[i]} \wedge \tilde{y}_i \neq \perp)$, and outputs $b' \wedge \neg\mathsf{BREAK}$ as the output of the experiment.

- H_2: The same as H_1, except for the following changes in steps 2 and 5.

Changes in H_2:

2. For a polynomial number of rounds $i = 1, 2, \ldots, q_1$, the adversary $\mathcal{A}_1$ may choose one of the following:
   (a) **Submit a $\mathsf{Gen}(n_i)$ query to $\mathcal{C}$:**
   $\mathcal{C}$ computes $(\sigma_i, y_i) \leftarrow \mathsf{Gen}(n_i)$, and responds with (σ_i, y_i).
   (b) **Submit a $\mathsf{Rep}(\tilde{\sigma}_i)$ query to $\mathcal{C}$:**
   $\mathcal{C}$ computes $\tilde{y}_i \leftarrow \mathsf{Rep}(\tilde{\sigma}_i)$. If $\tilde{\sigma}_i = \sigma_j$ for some $j < i$, return $\tilde{y}_i = y_j$. Otherwise, return $\perp$.
5. Again, for a polynomial number of rounds $i = q_1 + 1, q_1 + 2, \ldots, q_2$, the adversary $\mathcal{A}_1$ may choose one of the following:
   (a) **Submit a $\mathsf{Gen}(n_i)$ query to $\mathcal{C}$:**
   $\mathcal{C}$ computes $(\sigma_i, y_i) \leftarrow \mathsf{Gen}(n_i)$, and responds with (σ_i, r_i).
   (b) **Submit a $\mathsf{Rep}(\tilde{\sigma}_i)$ query to $\mathcal{C}$:**
   $\mathcal{C}$ computes $\tilde{y}_i \leftarrow \mathsf{Rep}(\tilde{\sigma}_i)$. If $\tilde{\sigma}_i = \sigma_j$ for some $j < i$, return $\tilde{y}_i = y_j$. Otherwise, return $\perp$.

- H_3: The same as H_2, except that in step 4d, instead of sending $(m_0 \oplus r, \sigma, z)$, now send (u, σ, z), where u is a uniform $|m_0|$-bit string.
- H_4: The same as H_3, except that in step 4d, revert back to sending $(m_1 \oplus r, \sigma, z)$.
- H_5: Revert the changes made in H_2. This is the same as $\mathsf{ExptEvltPriv}^{\mathsf{HELP},\Pi'}_{\mathcal{C},\mathcal{A},1}$ apart from the same adjustments highlighted in H_1.

First, we show that the outputs of H_1 and H_2 (W.L.O.G. also H_4 and H_5) are identically distributed. First, note that the only differences between H_2 and H_1 are how Rep queries are answered. Further, for a Rep query on $\tilde{\sigma}_i$, notice that the answer is y_i in both hybrids if there exists some $j < i$ such that $\tilde{\sigma}_i = \sigma_j$. In the other case, if $\tilde{\sigma}_i \neq \sigma_j$ for all $j < i$, then in H_2, $\mathcal{C}$ always responds $\perp$ (though it does compute $\tilde{y}_i$ anyway). In H_1, the response may or may not be $\perp$. If for all such Rep queries ($\tilde{\sigma}_i \neq \sigma_j$ for all $j < i$) in H_1, the answers are all $\perp$, then the adversary's view in H_1 and H_2 are identical, and hence outputs the same bit b'. If there exists such a Rep query with non-$\perp$ answer in H_1, the adversary's view might be different, causing b' to have a different distribution. But notice that in that case, BREAK is 1, so the output of the experiment is always 0 regardless of the adversary's output b'. Therefore, the outputs of H_1 and H_2 are identically distributed.

Next, we show that, the outputs of H_2 and H_3 are statistically close (and similarly H_3 and H_4). Concretely, we show that no unbounded distinguisher $\mathcal{D}$ can distinguish between the outputs of H_2 and H_3 with probability more than ξ. We argue this by reduction to ξ-privacy. We show how to use such a distinguisher $\mathcal{D}$ to build an adversary $\mathcal{B}$ for the privacy game. Notice that $\mathcal{B}$ needs to simulate the interactions between $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and $\mathcal{C}$ for H_2 and H_3 in order to produce the outputs of H_2 and H_3. $\mathcal{B}$ performs the simulation in the following manner, by running $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ "in its head" and pretending to be the challenger $\mathcal{C}$:

1. Runs the code of $\mathcal{A}_1$:
   (a) Whenever $\mathcal{A}_1$ submits a $\mathsf{Gen}(n_i)$ query to $\mathcal{C}$, forward it to the $\mathsf{Gen}(\cdot)$ oracle, and correspondingly the response back to $\mathcal{A}_1$.
   (b) Whenever $\mathcal{A}_1$ submits a $\mathsf{Rep}(\tilde{\sigma}_i)$ query to $\mathcal{C}$, check if there exists $j < i$ such that $\tilde{\sigma}_i = \sigma_j$. If exists, respond with $\tilde{y}_i = y_j$. Otherwise, respond with $\perp$.

(c) When $\mathcal{A}_1$ sends the challenge messages m_0 and m_1 to $\mathcal{C}$, $\mathcal{B}$ outputs $|m_0|$ and its current state in the privacy game. It then gets reactivated with the same state and receives r_b, σ, z. Then $\mathcal{B}$ run the message transmission protocol Π' to send $(r_b \oplus m_0, \sigma, z)$, which causes a certain leakage to $\mathcal{A}_1$.

2. Runs the code of $\mathcal{A}_2$ on $\mathsf{view}(\mathcal{A}_1)$ as input, when $\mathcal{A}_2$ outputs the bit b', produce the output of the experiment as just b'. Notice the difference from $b' \wedge \neg\mathsf{BREAK}$, as $\mathcal{B}$ cannot compute BREAK without access to the Rep oracle.

3. Give the output of the experiment to $\mathcal{D}$. If $\mathcal{D}$ outputs it receives the output from H_2, $\mathcal{B}$ output 0. Otherwise, $\mathcal{B}$ output 1.

Notice that the above simulated transcript by $\mathcal{B}$ is only correct if $\mathsf{BREAK} = 0$. However, if $\mathsf{BREAK} = 1$, the outputs of H_2 and H_3 are always 0, hence trivially no distinguisher can distinguish between them. In the cases where $\mathsf{BREAK} = 0$, if $\mathcal{D}$ successfully distinguishes H_2 and H_3, then the adversary $\mathcal{B}$ above successfully distinguishes between real or random in the privacy game. Therefore, we have shown that no unbounded distinguisher can distinguish between the outputs of H_2 and H_3 (also H_3 and H_4) with probability more than $\frac{1}{2} + \Pr[\neg\mathsf{BREAK}] \cdot \xi$, i.e. $|\Pr[H_2(\cdot) = 1] - \Pr[H_3(\cdot) = 1]| \leq 2 \cdot \Pr[\neg\mathsf{BREAK}] \cdot \xi \leq 2\xi$.

Bringing the five hybrids together, we have

$$\begin{aligned}&\Big|\Pr\Big[\mathsf{ExptEvltPriv}_{\mathcal{C},\mathcal{A},0}^{\mathsf{HELP},\Pi'}(N,\lambda) = 1 \wedge \neg\mathsf{BREAK}_0\Big]\\ &\quad - \Pr\Big[\mathsf{ExptEvltPriv}_{\mathcal{C},\mathcal{A},1}^{\mathsf{HELP},\Pi'}(N,\lambda) = 1 \wedge \neg\mathsf{BREAK}_1\Big]\Big|\\ &\leq |\Pr[H_2(\cdot) = 1] - \Pr[H_3(\cdot) = 1]| + |\Pr[H_3(\cdot) = 1] - \Pr[H_4(\cdot) = 1]|\\ &\leq 4\xi.\end{aligned}$$

□

Combining Lemma 1, Theorem 3, and the above Lemma 2 finishes the proof for Theorem 5.

□

Implications for HELP. This section showcases how one can use a HELP instance to elevate *any* message transmission protocol with computational privacy to *everlasting privacy.* This presents new paths towards building everlasting-secure protocols. Instead of building the schemes directly from information-theoretic assumptions, one can build a scheme that is secure only against computational adversaries, and then compose the construction with HELP to obtain everlasting security. While we only proved the case of everlasting privacy for message transmission schemes, we believe many other applications are possible, such as key exchange protocols. We leave these directions as interesting open questions for the reader.

6 Syndrome Resilient Functions

The goal of this section is to introduce the notion of syndrome resilient functions, which will be helpful for the multi-server HELP setting, but could be of independent interest. At a high level, this combines the error correction of syndrome decoding with the privacy guarantees of so called resilient functions [CGH+85]. Note, we use the same notation for dist as in Equation 1.

Definition 19 (Syndrome Resilient Function)**.** Let Σ be some alphabet, let $\tau = (t, t_a, t_f, t_p)$ be natural numbers, and let $\Delta, k \in \mathbb{N}$. Let $\mathsf{SRF} = (\mathsf{Eval}, \mathsf{Rec})$ for $\mathsf{Eval} : \Sigma^t \to \Sigma^\Delta \times \Sigma^k$ and $\mathsf{Rec} : (\Sigma \cup \bot)^t \times \Sigma^\Delta \to (\Sigma^k \cup \bot)$ for failure symbol $\bot$. Let $\mathsf{SRF} = (\mathsf{Eval}, \mathsf{Rec})$, and denote $\mathsf{Eval}(y) = (z, w)$, $\mathsf{Rec}(\tilde{y}, z) = \tilde{w}$ for $y \in \Sigma^t$, $\tilde{y} \in (\Sigma \cup \bot)^t$. We say SRF is a (τ, Δ, k)-syndrome resilient function if:

1. If $\mathsf{dist}(y, \tilde{y}) \leq (t_a, t_f)$, then $\mathsf{Rec}(z, \tilde{y}) = w$ for $\mathsf{Eval}(y) = (z, w)$.

2. For all subsets $BAD \subseteq [t]$ such that $|BAD| = t_p$, and all values y_i^* for $i \in BAD$, define random variable Y_i for all $i \in [t]$ as follows:

$$Y_i = \begin{cases} y_i^* & i \in BAD \\ U_\Sigma & i \notin BAD \end{cases}$$

   Then, if $(Z, W) = \mathsf{Eval}(Y_1, \ldots, Y_t)$, we require that $(Z, W) \equiv (Z, (U_\Sigma)^k)$.

Essentially, the first requirement of SRFs require that the recovery is error-correcting for all codewords with at most t_a adversarially-chosen points and at most t_f points returning $\bot$. The second ensures perfect secrecy for the recovered word even given t_p known points and the helper word z.

Construction. We will show that a Vandermonde matrix suffices for building an SRF. This is natural, as the notion of SRFs draws inspiration from *syndrome* decoding of Reed-Solomon codes [RS60]. For the sake of brevity, we introduce the minimal amount of terminology for this that we need to describe SRFs and our construction.

Definition 20 (Codes, Minimum Distance, Generator Matrix)**.** Let Σ be a finite alphabet, and let C be a subset of Σ^n (that is, a set of length n words in the alphabet). Then, we say C is a code over Σ, and the minimum distance of C is defined as the minimum Hamming distance between elements of C.

Further, if we let $\mathcal{M}$ be some message space, then we call $\mathbf{A} : \mathcal{M} \to C$ an encoding of $\mathcal{M}$ for code C. When $\mathbf{A}$ is a matrix, we may also call it the generator matrix of the code C.

The Vandermonde matrix is a well-known generator matrix. We describe it below, along with the useful properties it possesses for us.

Definition 21 (Vandermonde Matrix)**.** Let $\mathbb{F}$ be a field, and let $t, m \in \mathbb{N}$. Let $x_1, \ldots, x_t \in \mathbb{F}$ be arbitrary distinct field elements. Then, we define the $m \times t$ Vandermonde matrix, denoted $V_m = V_m(x_1, \ldots, x_t)$, as:

$$V_m = \begin{pmatrix} 1 & 1 & \ldots & 1 \\ x_1 & x_2 & \ldots & x_t \\ x_1^2 & x_2^2 & \ldots & x_t^2 \\ \ldots & & & \ldots \\ x_1^{m-1} & x_2^{m-1} & \ldots & x_t^{m-1} \end{pmatrix}.$$

The Vandermonde matrix allows us to do polynomial interpolation on the points $x_1, \ldots, x_t$ by multiplying a given coefficient (row) vector by V_m. This allows the Vandermonde matrix to have some incredibly useful properties in coding theory. In particular, we will take advantage of two well-known properties of the Vandermonde matrix:

1. It is a generator matrix of a Reed-Solomon code [RS60].

2. It is a parity check matrix for the generalized Reed-Solomon code.

In particular, the former property give us a simple encoding procedure for our SRF construction. The second property will give us the needed error correction for our decoding procedure. That is, given $y, \tilde{y} \in \mathbb{F}^t$ where $\tilde{y}$ and y have distance less than the minimum distance of the implicit Reed-Solomon code, there is a procedure $\mathsf{Decode}(\tilde{y}, V_m y)$ which returns y. Importantly, this minimum distance plays nicely with our notion of distance before; that is, this minimum distance is exactly $\mathsf{dist}(y, \tilde{y})$ in Equation 1.

As an aside, $V_m y$ here is called the syndrome of y, and this process is known as syndrome decoding, owing to the name SRF. With all this in mind, we are ready to present our construction of SRF.

Construction 6.1. *Let $\Sigma = \mathbb{F}$, $|\mathbb{F}| \geq t$, $\Delta = 2t_a + t_f$, and let V_m be the $m \times t$ Vandermonde matrix, where $m = t - t_p$. Define* $\mathsf{Eval}, \mathsf{Rec}$ *as follows:*

- $\mathsf{Eval}(y)$*: Let $V_m y = z||w$, where z is the first Δ entries of $V_m y$ and w is the remaining $t - t_p - \Delta$ entries. Output (z, w).*
- $\mathsf{Rec}(\tilde{y}, z)$*: Run* $\mathsf{Decode}(\tilde{y}, z)$ *as above, receiving $y \in \{0,1\}^n \cup \perp$. If $y \neq \perp$, let $V_m y = z||w$ as in* Eval*, and output w.*

The notion of SRFs also draws on a primitive known as *resilient functions* (and their relaxations, exposure-resilient functions [CDH+00]). The notion of resilience is as such relevant to the proof of our SRF construction, so we include it here.

Definition 22 (Resilient Function)**.** A deterministic polynomial time computable function $\mathsf{rf} : \{0,1\}^t \to \{0,1\}^k$ is said to be an ℓ-resilient function if for all subsets $L \subseteq [t]$ such that $|L| = t - \ell$ and all $r \in \{0,1\}^{t-k}$, we have

$$\langle \mathsf{rf}(U|_{t:L|_r}) \rangle = \langle U_k \rangle,$$

where $U|_{t:L|_r}$ denotes the uniform distribution over t-bit strings where the L-th positions are equal to r.

We now have the vocabulary needed to prove that Construction 6.1 is an SRF.

Theorem 6. *For $\tau = (t, t_a, t_f, t_p)$ satisfying $|\Sigma| \geq t$, $t - t_p \geq 2t_a + t_f$, define $\Delta = 2t_a + t_f$. Then, Construction 6.1 is a $(\tau, \Delta, (t - t_p - \Delta))$-SRF.*

Proof. We prove each property separately.

Property 1. Let $Y, \tilde{Y} \in \mathbb{F}^t$ such that $\mathsf{dist}(Y, \tilde{Y}) \leq (t_a, t_f)$. By definition, this means $\tilde{Y}$ has at most t_f erasures and otherwise differs from Y in at most t_a positions. Because the minimum distance of the Reed Solomon code which V_m generates is $\Delta = 2t_a + t_f$, we see Decode can correct the errors and erasures. This yields the first property, as $\mathsf{Rec}(\tilde{y}, z)$ simply runs Decode.

Property 2. Let $BAD \subseteq [t]$ be a subset of size t_p, and let $y_i^* \in \mathbb{F}$ be arbitrary for $i \in BAD$. We define the random variable $Y = (Y_1, \ldots, Y_t)$ where each $Y_i = y_i^*$ if $i \in BAD$ and is uniformly sampled from $\mathbb{F}$ otherwise. We see from Theorem 1 of [CGH+85] that, since $t \leq 2^n - 1$, there exists a (t, t_p)-perfect exposure resilient function. In particular, Theorem 2 of [CGH+85] gives us that the generator matrix of the generalized Reed-Solomon with minimum distance $t - t_p + 1$ is this exposure resilient function. So, as long as $t - t_p \geq 2t_a + t_f$, V_m perfectly hides $t - t_p$ points from t_p exposures, satisfying the second property. □

Instantiating SRF. In the following section, we will use SRFs to process the received random bits received from each server. This will intuitively allow us to retrieve some certified (Definition 19 Property 1), private (Property 2) randomness from the n-bit samples provided by the t servers. Using Construction 6.1 with $\mathbb{F} = \{0,1\}^n$ in this setting will work, but the multiplications needed for the resulting $\mathsf{Eval}, \mathsf{Rec}$ will be over a very large field.

To make this more efficient, we note that we can make this more efficient by separating the retrieved n-bit samples into $n/\log t$ blocks of length $\log t$, applying Construction 6.1 to each of these. While we elide this discussion to Section 7.3, our practical construction will take this into account.

7 Distributed HELP

We generalize our model to the distributed setting, where now A and B may communicate with one of t servers $\mathcal{S}_1, \ldots, \mathcal{S}_t$. As we will see, this setting will be very similar to the single-server setting, but with minor changes to correctness and security. Crucially, we will rely on SRFs (Section 6) to achieve our strong correctness and distributed privacy guarantees. By satisfying these, though, all our results conveniently generalize, including everlasting privacy (Section 5).

7.1 Redefining Distributed Syntax and Correctness

In our generalization to $t \geq 1$ servers, we will allow some of the servers to be failing, adversarial, or non-private. While users will not know which of the servers will be faulty in some way, we assume we know reasonable upper bounds on each type of servers. These are defined below. We will use the following parameters for bounds on these faulty servers:

- t — the *total* number of servers;
- t_a — the number of *adversarially* chosen servers, which may reproduce arbitrarily different values from the original generated randomness;
- t_f — the number of *failing* servers, which may return $\perp$ upon calling Rep, even for a valid tag;
- t_p — the number of *public* servers, which the adversary has full view on (which may be distinct from the previous categories).

Additionally, we will use the notation $t_g = t - t_p$ to represent the "good" servers which the users have private channels with. We will also use the following notation to simplify definitions, as the above are used as parameters for many of the distributed syntax:

$$\tau := (t, t_a, t_f, t_p).$$

Syntax. In this setting, $\mathsf{Init}, \mathsf{Gen}$ and Rep are unchanged, since we want our servers to act independently. In fact, the servers need not know of how many other servers are participating, their identities, etc. However, to simplify the notation, we will use a subscript to differentiate servers from each other. We stress this is only done when describing the correctness and security of the system, but each server is not aware of this index. Moreover, servers need not care about the tuple τ described above, as this is only relevant for the users of the system. Thus, new server syntax is as follows:

- $\mathsf{Init}_i(N, 1^\lambda) \to (X_i, \mu_i)$: The initialization algorithm run by server $\mathcal{S}_i$. We denote $|X_i| =: \tilde{N} \geq N$. When initializing all the servers, we use the shortcut $\mathsf{Init}(N, 1^\lambda) := \{\mathsf{Init}_i(N, 1^\lambda)\}_{i \in [t]}$.

- $\mathsf{Gen}_i(n, (X_i, \mu_i)) \to (\sigma_i, y_i, \mu_i)$: The pad generation query for server $\mathcal{S}_i$ for n bits of randomness y_i and tag σ_i.
- $\mathsf{Rep}_i(\tilde{\sigma}_i, (X_i, \mu_i)) \to \tilde{y}_i$: The pad reproduction query for server $\mathcal{S}_i$, which reproduces $\tilde{y}_i$ from tag $\tilde{\sigma}_i$.

As in the single server setting, we will elide mentions of X_i and μ_i in Gen_i and Rep_i, when obvious.

Turning to the user, they also run the same algorithms $\mathsf{PadLength}, \mathsf{Auth}$ and Ver, but now these algorithms also take the tuple $\tau = (t, t_a, t_f, t_p)$ above. For example, the length calculation algorithm becomes:

- $\mathsf{PadLength}(\ell, \tau, 1^\lambda) \to n$.

Intuitively, since the user will get n bits of randomness from t servers, we could hope to extract close to $\ell \approx tn$ bit. However, since only t_g out of t servers provide private randomness, the correct expectation is to have $\ell \approx t_g n$. Equivalently, if our randomness extraction procedure is good, we will manage to set $n \approx \ell/t_g$. But the exact formula will also depend on the security parameter to ensure user integrity later.

Algorithms Auth and Ver are also similar. First, instead of taking a single n-bit pad y, now they take t such pads $y_1, \ldots, y_t$. To simplify notation, we will still denote the vector of these pads by $y := (y_1, \ldots, y_t)$. And similarly for the vector of tags $\sigma := (\sigma_1, \ldots, \sigma_t)$ returned by t called to Gen. With these conventions, the new syntax is very similar to the earlier syntax:

- $\mathsf{Auth}(\sigma, y, \tau, 1^\lambda) \to (z, r)$. Transforms the tags $\sigma = (\sigma_1, \ldots, \sigma_t)$ and the pads $y = (y_1, \ldots, y_t)$ to a (single) ℓ-bit key r and a (single) helper string z.
- $\mathsf{Ver}(z, \tilde{y}, \tau) \to \tilde{r}$. Checks the helper string z against (recovered) pads $\tilde{y} = (\tilde{y}_1, \ldots, \tilde{y}_t)$, and outputs a key $\tilde{r} \in \{0,1\}^\ell \cup \{\perp\}$.
- When the value τ is clear from context, we will sometimes omit it as an explicit input to $\mathsf{Auth}, \mathsf{Ver}$.

The only additional twist, required by our new distributed Correctness below, comes from the fact that some t_f servers might be unavailable when the user calls Rep for these servers. Thus, in the verification algorithm Ver we allow up to t_f values $\tilde{y}_i$ to be equal to $\perp$.

Correctness. Recall, in the single-server case, we separately defined server-correctness (Definition 5) and user-correctness (Definition 6), which immediately implied overall correctness of the HELP scheme. We do the same here. First, server-correctness does not change, as each server runs independently without knowing about the other servers.

For user-correctness, we need the following definition. Given a t-value vector $y = (y_1, \ldots, y_t)$ over alphabet $\{0,1\}^n$ (so each $y_i \in \{0,1\}^n$), and a t-value vector $\tilde{y} = (\tilde{y}_1, \ldots, \tilde{y}_t)$ over alphabet $\{0,1\}^n \cup \{\perp\}$ (so each $\tilde{y}_i \in \{0,1\}^n \cup \{\perp\}$), we say that

$$\mathsf{dist}(y, \tilde{y}) \le (t_a, t_f) \tag{1}$$

if (a) $|\{j : \tilde{y}_j \notin \{y_j, \perp\}| \le t_a$; and (b) $|\{j : \tilde{y}_j = \perp\}| \le t_f$. With this in mind, the user-correctness is defined below:

Definition 23. HELP satisfies *user-correctness* for a given $\tau = (t, t_a, t_f, t_p)$ if, for all $y \in (\{0,1\}^n)^t$ and $\tilde{y} \in (\{0,1\}^n \cup \{\perp\})^t$ with $\mathsf{dist}(y, \tilde{y}) \le (t_a, t_f)$, and all t-value vectors σ we have:

$$\Pr\left[\ \tilde{r} = r \ \middle|\ (z, r) \leftarrow \mathsf{Auth}(\sigma, y, \tau, 1^\lambda),\ \ \tilde{r} \leftarrow \mathsf{Ver}(z, \tilde{y}, \tau)\ \right] = 1.$$

We can also show that server-correctness and (distributed) user-correctness imply the overall (distributed) correctness of the scheme. However, we will need to explicitly introduce a (potentially unbounded) attacker $\mathcal{A}$ which is allowed to arbitrarily modify the correct t-tuple y of Rep-responses, into a corrupted t-tuple $\tilde{y}$ satisfying $\mathsf{dist}(y, \tilde{y}) \leq (t_a, t_f)$. The overall correctness will still ensure that the users output correct derived key $\tilde{r} = r$. For simplicity of notation, we omit this straightforward implication.

7.2 Redefining Distributed Security

Recall, security of HELP consists of three components: *server authentication*, *user integrity* and *privacy*. We now show how to extend them from the single-server setting (see Section 3.2) to the distributed setting.

Distributed Server Authentication. Since each of the servers run independently, and have the same syntax, we simply require that the (single-server) server authentication given in Definition 8 must hold for all t servers.[9]

Distributed User Integrity. We make only a couple of syntactic changes to single-server User integrity in Definition 9. Instead, we model a single adversary controlling all t servers.

Definition 24. Let λ be the security parameter. We say that HELP satisfies δ-*user integrity* with $\delta = \mathtt{negl}(\lambda)$ for a given $\tau = (t, t_a, t_f, t_p)$, if for all PPT attackers $\mathcal{S}$, we have

$$\Pr\left[\ \tilde{r} \notin \{r, \perp\}\ \middle|\ \begin{array}{l} (st, \sigma, y) \leftarrow \mathcal{S}(1^\lambda); (z, r) \leftarrow \mathsf{Auth}(\sigma, y, \tau, 1^\lambda); \\ \tilde{y} \leftarrow \mathcal{S}(st, z, r); \tilde{r} = \mathsf{Ver}(z, \tilde{y}, \tau) \end{array} \right] \leq \delta.$$

Notice, σ, y and $\tilde{y}$ are now t-element vectors, and further $\tilde{y}$ may contain $\perp$ at some points (up to t_f). Intuitively, this models the attacker acting on behalf of the as the entire collective of the t servers, as Auth and Ver are over the entirety of the server responses. Of course, this also implies integrity when only some of the servers are malicious, and others are honest.

Distributed Privacy. This property involves the main difference from the single-server case (see Definition 10), as this time we deal with an attacker who compromised a certain number t_p of the servers, unbeknownst to the users. For simplicity of definition, we will assume a static attacker — i.e., the t_p bad servers are chosen at setup — but we believe that our results should work for active attackers as well.

For notation, let BAD be this set of compromised servers, so $|BAD| \leq t_p$. Finally, for a t-valued vector $X = (X_1, \ldots, X_t)$, we let $X\ |_{BAD} = \cup_{i \in BAD} X_i$.

Definition 25. Let λ be the security parameter, and let $\tau = (t, t_a, t_f, t_p)$. We say that HELP satisfies ξ-privacy for $\xi = \mathtt{negl}(\lambda)$, if for all subsets $BAD \subseteq [t]$ such that $|BAD| \leq t_p$, all *unbounded* adversaries $\mathcal{A}$ with oracle access to $\{\mathsf{Gen}_i\}_{i\in[t]}$, any value of N, after we run $\mathsf{Init}(N, \tau, 1^\lambda)$ to initialize $\{(X_i, \mu_i)\}_{i\in[t]}$ and sample a random bit $b \overset{\$}{\leftarrow} \{0,1\}$, we have

$$\Pr\left[\ b' = b\ \middle|\ \begin{array}{l} (\ell, st) \leftarrow \mathcal{A}^{\mathsf{Gen}}(X\ |_{BAD}, N, 1^\lambda);\ n \leftarrow \mathsf{PadLength}(\ell, \tau, 1^\lambda); \\ \forall i \in [t], (\sigma_i, y_i) \leftarrow \mathsf{Gen}_i(n); \\ y = \{y_i\},\ \sigma = \{\sigma_i\},\ (z, r_0) \leftarrow \mathsf{Auth}(\sigma, y, \tau, 1^\lambda); \\ r_1 \overset{\$}{\leftarrow} \{0,1\}^\ell;\ b' \leftarrow \mathcal{A}^{\mathsf{Gen}}(st, \sigma, z, r_b) \end{array} \right] \leq \frac{1}{2} + \xi.$$

[9] Technically, we can let the attacker interact with all t servers, and dynamically choose the one to attack. But a simple hybrid argument shows that this definition is easily implied by satisfying t individual server authentication definitions.

Essentially, the adversary is assumed to have complete knowledge of all of the t_p compromised servers, including their internal randomness $X\mid_{BAD}$.

Remark. For simplicity of exposition, in our privacy definition above we assumed that the t_p compromised servers are acting honestly otherwise. We could have considered them fully byzantine, by allowing the attacker $\mathcal{A}$ to also control the oracle Gen_i for $i \in BAD$, used by the challenger to compute the values y_i. Indeed, it is not hard to see that our subsequent scheme in Section 7.3 would satisfy this notion. But we decided to stay with a simpler-to-define privacy notion instead.

7.3 Distributed HELP Construction

We present our construction of HELP with t servers. At a high level, we will use an SRF to process the communication with the t servers in order to approximate the communication of single-server HELP. In this way, the error correcting of SRFs will allow the user to reconstruct even given t_a incorrect server responses and t_f failures, and the perfect secrecy of the SRF will allow us to achieve privacy even when the adversary sees some of the servers' communications.

Construction 7.1. *For security parameter λ and parameters $\tau = (t, t_a, t_f, t_p)$, let $\mathsf{SRF} = (\mathsf{Eval}, \mathsf{Rec})$ be a $(\tau, 2t_a + t_f, n)$-SRF, let $\mathcal{E}_\lambda = \{\mathsf{EH}\}$ be an extractor-hash family with helper length ℓ_{help}, and let $\mathsf{Mac.Tag} : \{0,1\}^* \times \{0,1\}^{\ell_{MAC}} \to \{0,1\}^{\ell_{tag}}$ be a computational MAC. Then, we will define*

$$\mathsf{HELP} = (\{\mathsf{Init}_i, \mathsf{Gen}_i, \mathsf{Rep}_i\}_{i\in[t]}, \mathsf{PadLength}, \mathsf{Auth}, \mathsf{Ver})$$

as so, starting first with the initialization and length:

- $\mathsf{Init}_i(N, \tau, 1^\lambda)$*: On input N, set $\tilde{N} = N + \ell_{MAC}$ and sample $X_i \leftarrow \{0,1\}^{\tilde{N}}$ uniformly at random. Parse $X_i = (k_i, X_{i,pad})$, where $|k| = \ell_{MAC}$. Set $\mu_i = (1, N)$.*
- $\mathsf{PadLength}(\ell, \tau, 1^\lambda)$*: On input $(\ell, \tau, 1^\lambda)$, output $n = (\ell + \ell_{help})/(t - t_p - 2t_a - t_f)$.*

With these in mind, we define how each server accepts and responds to $\mathsf{Gen}, \mathsf{Rep}$ *queries:*

- $\mathsf{Gen}_i(n)$*: Let $\mu_i = (\mathsf{index}, N)$. If $\mathsf{index} + n > N$, return $\perp$. Else, define $y_i = X_{i,pad}[\mathsf{index}, \ldots, \mathsf{index} + n - 1]$. Set $\sigma_i = (\mathsf{Mac.Tag}_{k_i}(\mathsf{index}, n), \mathsf{index}, n)$. Then, output (y_i, σ_i) and set $\mu = (\mathsf{index} + n, N)$.*
- $\mathsf{Rep}_i(\sigma)$ *: Parse $\sigma = (\mathsf{tag}, \mathsf{index}, n)$. Then, return $\tilde{y}_i = X_{i,pad}[\mathsf{index}, \ldots, \mathsf{index} + n - 1]$ if and only if $\mathsf{tag} = \mathsf{Mac.Tag}_{k_i}(\mathsf{index}, n)$.*

Finally, to parse these queries for pads, we define:

- $\mathsf{Auth}(\sigma, y, \tau)$ *: On input (σ, y), compute $\mathsf{Eval}(y) = (z_{srf}, w)$ and $\mathsf{EH}(w) = (z_{EH}, r)$. Set $z = (z_{srf}, z_{EH})$ and output (z, r).*
- $\mathsf{Ver}(z, \tilde{y}, \tau)$ *: On input $(z, \tilde{y})$, parse $z = (z_{srf}, z_{EH})$. Set $\tilde{w} = \mathsf{Rec}(\tilde{y}, z_{srf})$, and set $(\tilde{z}_{EH}, \tilde{r}) \leftarrow \mathsf{EH}(\tilde{w})$. Output $\tilde{r}$ if and only if $z_{EH} = \tilde{z}_{EH}$.*

As mentioned when describing the syntax, we note the above construction is for ultimately extracting $t \times \ell$ bits of randomness from the servers.

Theorem 7. *Let $t, t_p, t_f, t_a \in \mathbb{N}$ such that $t - t_p > t_f + 2t_a$, and let λ be the security parameter. If* SRF *is a $(\tau, 2t_a + t_f, n)$-SRF, $\mathcal{E}_\lambda$ is an extractor-hash and* $\mathsf{Mac.Tag}$ *is a computational MAC, then Construction 7.1 is a distributed* HELP.

Proof. Server-Correctness follows from Theorem 2, as $\mathsf{Init}_i, \mathsf{Gen}_i, \mathsf{Rep}_i$ are unchanged from the single-server construction. We prove user-correctness, server authentication, user integrity, and privacy.

User-Correctness: Let $y = (y_1, \ldots, y_t)$ and $\tilde{y} = (\tilde{y}_1, \ldots, \tilde{y}_t)$ be arbitrary satisfying $\mathsf{dist}(y, \tilde{y}) \leq (t_a, t_f)$. By construction, we have that $\mathsf{Auth}(\sigma, y, \tau) = (z, r)$ satisfying $\mathsf{Eval}(y) = (z_{srf}, w)$, $\mathsf{EH}(w) = (z_{EH}, r)$, and $z = (z_{srf}, z_{EH})$. We also have by construction that $\mathsf{Ver}(z, \tilde{y}, \tau) = \tilde{r}$ satisfying $z = (z_{srf}, z_{EH})$, $\tilde{w} = \mathsf{Rec}(\tilde{y}, z_{srf})$, $(\tilde{z}_{EH}, \tilde{r}) = \mathsf{EH}(\tilde{w})$, and $r = \tilde{r}$ if and only if $z_{EH} = \tilde{z}_{EH}$.

By the first property of SRF, we have that $\tilde{w} = \mathsf{Rec}(z_{srf}, \tilde{y}) = w$. So, we have that

$$(\tilde{z}_{EH}, \tilde{r}) = \mathsf{EH}(\tilde{w}) = \mathsf{EH}(w) = (z_{EH}, r).$$

We conclude that this implies that $\mathsf{Ver}(z, \tilde{y}, \tau) = \tilde{r} = r$ always.

Server Authentication: Note that server authentication only relies on queries which are the same as in Construction 4.2. Unsurprisingly, then, the proof here will follow straightforwardly from as in the single server case. Suppose for some λ, N there exists a PPT adversary $\mathcal{A}$ with oracle access to $\mathsf{Gen}_i, \mathsf{Rep}_i, \mathsf{Gen}_i^*$ for all $i \in [t]$ and some polynomial $p(\lambda)$ such that:

$$\Pr\left[\ \mathsf{Rep}_j(\sigma^*) \neq \perp \ \middle|\ (j, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Gen}_i, \mathsf{Rep}_i, \mathsf{Gen}_i^*}(N, 1^\lambda)\ \right] > \frac{1}{p(\lambda)},$$

where σ^* was not queried by $\mathcal{A}$. We construct $\mathcal{B}$ that can forge message authentication codes for $\mathsf{Mac.Tag}$. At the start, $\mathcal{B}^{\mathsf{Mac.Tag}_k}$ will choose a random point $i \in [t]$, which will serve as the server index for which it injects $\mathsf{Mac.Tag}_k$. For all points $j \in [t] \setminus \{i\}$, $\mathcal{B}$ will act as in the regular challenge, running $\mathsf{Init}_j(N, \tau, 1^\lambda)$ and responding to $\mathsf{Gen}_j, \mathsf{Rep}_j$, and Gen_j^* queries normally. For server i, though, $\mathcal{B}$ will run instead as in the single-server case, sampling $X_{pad} \leftarrow \{0,1\}^N$ uniformly at random and setting $\mathsf{index} = 1$. With all this setup done, $\mathcal{B}$ will begin running $\mathcal{A}$, responding to $\mathsf{Gen}, \mathsf{Rep}, \mathsf{Gen}^*$ queries to servers in $[t] \setminus \{i\}$ normally.

For server i, $\mathcal{B}$ will perform essentially as in the reduction proof for single-server authentication, which we reproduce below (with minor syntactical changes):

- For $\mathsf{Gen}_i(n)$ queries, $\mathcal{B}$ will first check if $\mathsf{index} + n > N$. If so, return $\perp$ right away. Otherwise, $\mathcal{B}$ queries $\mathsf{Mac.Tag}_k$ with (index, n), receiving a tag tag. $\mathcal{B}$ sets $y = X[\mathsf{index}, \ldots, \mathsf{index} + n - 1]$ and $\sigma = (\mathsf{tag}, \mathsf{index}, n)$, updates $\mathsf{index} = \mathsf{index} + n$, returns and stores (y, σ).
- For $\mathsf{Gen}_i^*(n)$ queries, $\mathcal{B}$ completes the same process as above, except that it now queries/computes $\mathsf{Mac.Tag}_{k^*}$ instead, where k^* is a MAC key sampled by $\mathcal{B}$ itself. $\mathcal{B}$ also only returns y as the response.
- For $\mathsf{Rep}_i(\tilde{\sigma})$ queries, $\mathcal{B}$ will simply respond with one of the stored y if and only if $\tilde{\sigma}$ is equal to the corresponding σ stored.

When $\mathcal{A}$ completes and outputs (j, σ^*), $\mathcal{B}$ aborts if $j \neq i$. Otherwise, $\mathcal{B}$ parses $\sigma^* = (\mathsf{tag}, \mathsf{index}, n)$ and outputs that tag is a valid $\mathsf{Mac.Tag}$ tag for (index, n).

As argued in the single-server case, $\mathcal{B}$ wins the MAC forgery game if $\mathcal{A}$ wins the server authentication game and $j = i$. Notice that since i is sampled uniformly and independent from the view of $\mathcal{A}$, $j = i$ with probability $1/t$. So if $\mathcal{A}$ succeeds with probability $1/p(\lambda)$, $\mathcal{B}$ would succeed with probability $1/tp(\lambda)$, which is more than negligible. Hence, construction 7.1 satisfies server authentication.

User Integrity: As before, we will rely on the collision resistance property of extractor-hash to ensure user integrity. Suppose for some λ there exists a PPT adversarial server ensemble $\mathcal{S}$ and some polynomial $p(\lambda)$ such that

$$\Pr\left[\tilde{r} \notin \{r, \perp\} \ \middle|\ \begin{array}{l} (st, \sigma, y) \leftarrow \mathcal{S}(1^\lambda); (z, r) \leftarrow \mathsf{Auth}(\sigma, y, \tau, 1^\lambda); \\ \tilde{y} \leftarrow \mathcal{S}(st, z, r); \tilde{r} = \mathsf{Ver}(z, \tilde{y}, \tau) \end{array}\ \right] > \frac{1}{p(\lambda)}.$$

We will use this to break the collision resistance property of EH as so: To construct $\mathcal{A}$, simply run $\mathcal{S}(1^\lambda)$, receiving $(st, \sigma, y), \tilde{y}$ from $\mathcal{S}$ satisfying $(z, r) \leftarrow \mathsf{Auth}(\sigma, y, \tau, 1^\lambda)$ and $\tilde{r} = \mathsf{Ver}(z, \tilde{y}, \tau)$. Now, as $\mathcal{A}$ computes and returns $\mathsf{Auth}(\sigma, y, \tau, 1^\lambda)$, it parses $z = (z_{srf}, z_{EH})$. Let $w = \mathsf{Rec}(y, z_{srf})$ and $\tilde{w} = \mathsf{Rec}(\tilde{y}, z_{srf})$. From this, $\mathcal{A}$ outputs $(w, \tilde{w})$.

Notice that since $\mathcal{S}$ wins the user integrity game, we have $\tilde{r} \notin \{r, \perp\}$. This means for $(\tilde{z}_{EH}, \tilde{r}) \leftarrow \mathsf{EH}(\tilde{w})$, we have $\tilde{z}_{EH} = z_{EH}$ and $\tilde{r} \neq r$, where as $(z_{EH}, r) \leftarrow \mathsf{EH}(w)$. So $(w, \tilde{w})$ is indeed a collision for the extractor-hash.

Distributed Privacy: It suffices to show the distribution of b' is the same for $b = 0, 1$. We will prove this via a series of (short) hybrid arguments.

H_1 : The game as described above when $b = 0$. So, $\mathcal{A}$ is initially given $X \mid_{BAD}$ and outputs (ℓ, st). Then, $\mathcal{A}$ is given $(z, r_0) \leftarrow \mathsf{Auth}(\sigma, y, \tau, 1^\lambda)$ alongside st and tag σ and outputs bit b'. By construction, $\mathsf{Auth}(\sigma, y, \tau, 1^\lambda)$ is defined as computing $\mathsf{Eval}(y) = (z_{srf}, w)$ and $\mathsf{EH}(w) = (z_{EH}, r)$, and outputting $(z = (z_{srf}, z_{EH}), r)$.

H_2 : The same game as H_1, but replace w with a uniformly sampled $\tilde{w} \sim \{0,1\}^n$.

H_3 : The same game as H_2, but replace r_0 with a uniformly sampled $\tilde{r} \sim \{0,1\}^\ell$.

H_4 : The original game when $b = 1$.

Note that $\mathcal{A}$ is an unbounded adversary, so we need statistical closeness of our hybrids. We now prove this for each pair.

$H_1 = H_2$**:** This comes directly from the second property of SRFs. That is, by construction of Init, we see $X_i \sim \{0,1\}^N$ for $i \notin BAD$. We also have that $\sigma = \{\sigma_i\}_{i \in [t]}$ is independent of all X_i (and so all y_i), so we have $\mathsf{Eval}(y) = (z_{srf}, w) = (z_{srf}, \tilde{w})$ as distributions.

$H_2 \approx H_3$**:** This follows analogously from the proof for single-server privacy that utilizes the extraction property of EH, with the addition that z_{srf} can just be sampled independently as a subroutine.

$H_3 = H_4$**:** Note that in H_3 the view of the adversary is now $X \mid_{BAD}$, st, $\sigma = \{\sigma_i\}_{i \in [t]}$, $z = (z_{srf}, \tilde{z}_{EH})$, and $\tilde{r} \sim \{0,1\}^\ell$, where $(\tilde{z}_{EH}, \tilde{r}) = \mathsf{EH}(\tilde{w})$ for $\tilde{w} \sim \{0,1\}^n$. By the same reasoning as in the first hybrid argument, we may replace $\tilde{w}$ with w as $\mathsf{Eval}(y) = (z_{srf}, w)$ and consequently switch $\tilde{z}_{EH}$ back to z_{EH}. It is clear from this that this is now exactly distributed as in H_4 (the original game).

The compilation of these hybrids shows that the output bit b' is statistically close in the game where the input bit b is 0 or 1, so no unbounded adversary can distinguish with non-negligible advantage. □

Extending to Everlasting Privacy. Notice that construction 5.1 for composing with a message transmission protocol in section 5 works with distributed HELP analogously. We highlight the key modifications needed below:

- In Definition 16, the adversary $\mathcal{A}$ will have access to $\mathsf{Gen}_i(\cdot)$ and $\mathsf{Rep}_i(\cdot)$ queries for *each of the* HELP *servers*, and when $\mathcal{C}$ sends a message, it will use the distributed HELP servers accordingly. Similar to the plain distributed server authentication definition, $\mathcal{A}$ wins if it breaks server authentication of any single server.
- In the proof for Theorem 3, in the reduction for H_2, $\mathcal{A}'$ would forward the oracle queries to the corresponding server. H_2 now reduces to Distributed Server Authentication instead.
- In Definition 17, the adversarial server $\mathcal{S}$ would now have control of all the t HELP servers, as similar to how distributed user integrity is defined in Definition 24.

- In the proof for Theorem 4, $\mathcal{S}$ and $\mathcal{S}'$ now correspond to all the t servers, and the produced y and σ are now vectors. It now also reduces to Distributed User Integrity (Definition 24).
- In Definition 18, the first stage adversary $\mathcal{A}_1$ will have access to $\mathsf{Gen}_i(\cdot)$ and $\mathsf{Rep}_i(\cdot)$ queries for *each of the* HELP *servers.*
- In the proof for Theorem 5, the BREAK predicates now correspond to breaking server authentication for *any* server. The probability of these predicates are thus bounded by the composed distributed server authentication error.
- In Lemma 2 and the proof of it, use the definition of Distributed Privacy (Definition 25) instead. The reduction proof should follow by reducing to Distributed Privacy, accordingly.

References

[ABB$^+$14] Romain Alléaume, Cyril Branciard, Jan Bouda, Thierry Debuisschert, Mehrdad Dianati, Nicolas Gisin, Mark Godfrey, Philippe Grangier, Thomas Länger, Norbert Lütkenhaus, et al. Using quantum key distribution for cryptographic purposes: a survey. *Theoretical Computer Science*, 560:62–81, 2014. doi:10.1016/j.tcs.2014.09.018.

[ADR02] Y. Aumann, Yan Zong Ding, and M.O. Rabin. Everlasting security in the bounded storage model. *IEEE Transactions on Information Theory*, 48(6):1668–1680, 2002. doi:10.1109/TIT.2002.1003845.

[ARML06] Romain Alléaume, François Roueff, Oliver Maurhart, and N Lutkenhaus. Architecture, security and topology of a global quantum key distribution network. In *2006 Digest of the LEOS Summer Topical Meetings*, pages 38–39. IEEE, 2006. doi:10.1109/LEOSST.2006.1694006.

[BB14] Charles H Bennett and Gilles Brassard. Quantum cryptography: Public key distribution and coin tossing. *Theoretical computer science*, 560:7–11, 2014. doi:10.1016/j.tcs.2014.05.025.

[BCEQ24] Chris Brzuska, Geoffroy Couteau, Christoph Egger, and Willy Quach. On bounded storage key agreement and one-way functions. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024, Part I*, volume 15364 of *LNCS*, pages 287–318. Springer, Cham, December 2024. doi:10.1007/978-3-031-78011-0_10.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988. doi:10.1145/62212.62213.

[BMV06] Johannes Buchmann, Alexander May, and Ulrich Vollmer. Perspectives for cryptographic long-term security. *Communications of the ACM*, 49(9):50–55, 2006. doi:10.1145/1151030.1151055.

[BPP05] H Bechmann-Pasquinucci and Andrea Pasquinucci. Quantum key distribution with trusted quantum relay. *arXiv preprint quant-ph/0505089*, 2005. doi:10.48550/arXiv.quant-ph/0505089.

[BZG+23] Riccardo Bassi, Qiaolun Zhang, Alberto Gatto, Massimo Tornatore, and Giacomo Verticale. Quantum key distribution with trusted relay using an etsi-compliant software-defined controller. In *2023 19th International Conference on the Design of Reliable Communication Networks (DRCN)*, pages 1–7. IEEE, 2023. doi:10.1109/DRCN57075.2023.10108347.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001. doi:10.1109/SFCS.2001.959888.

[CCH+22] Matthew Campagna, Craig Costello, Basil Hess, Aaron Hutchinson, Amir Jalali, Koray Karabina, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, et al. Supersingular isogeny key encapsulation, 2022. URL: https://csrc.nist.gov/csrc/media/Projects/post-quantum-cryptography/documents/round-4/submissions/SIKE-spec.pdf.

[CD23] Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 423–447. Springer, Cham, April 2023. doi:10.1007/978-3-031-30589-4_15.

[CDH+00] Ran Canetti, Yevgeniy Dodis, Shai Halevi, Eyal Kushilevitz, and Amit Sahai. Exposure-resilient functions and all-or-nothing transforms. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 453–469. Springer, Berlin, Heidelberg, May 2000. doi:10.1007/3-540-45539-6_33.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, Berlin, Heidelberg, August 2001. doi:10.1007/3-540-44647-8_2.

[CGH+85] Benny Chor, Oded Goldreich, Johan Hasted, Joel Freidmann, Steven Rudich, and Roman Smolensky. The bit extraction problem or t-resilient functions. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, pages 396–407, 1985. doi:10.1109/SFCS.1985.55.

[Che24] Yilei Chen. Quantum algorithms for lattice problems. Cryptology ePrint Archive, Report 2024/555, 2024. URL: https://eprint.iacr.org/2024/555.

[CZL+21] Yuan Cao, Yongli Zhao, Jun Li, Rui Lin, Jie Zhang, and Jiajia Chen. Hybrid trusted/untrusted relay-based quantum key distribution over optical backbone networks. *IEEE Journal on Selected Areas in Communications*, 39(9):2701–2718, 2021. doi:10.1109/JSAC.2021.3064662.

[Dam89] Ivan Damgård. A design principle for hash functions. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer, 1989. doi:10.1007/0-387-34805-0_39.

[DDWY93] Danny Dolev, Cynthia Dwork, Orli Waarts, and Moti Yung. Perfectly secure message transmission. *Journal of the ACM*, 40(1):17–47, January 1993. doi:10.1145/138027.138036.

[DM04] Stefan Dziembowski and Ueli M. Maurer. On generating the initial key in the bounded-storage model. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 126–137. Springer, Berlin, Heidelberg, May 2004. doi:10.1007/978-3-540-24676-3_8.

[Dod12] Yevgeniy Dodis. Shannon impossibility, revisited. In Adam Smith, editor, *ICITS 12*, volume 7412 of *LNCS*, pages 100–110. Springer, Berlin, Heidelberg, August 2012. doi:10.1007/978-3-642-32284-6_6.

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *SIAM Journal of Computing*, 2008. doi:10.1137/060651380.

[DP08] Yevgeniy Dodis and Prashant Puniya. Getting the best out of existing hash functions; or what if we are stuck with sha? In *Applied Cryptography and Network Security. ACNS 2008*, volume 5037. Springer, 2008. doi:10.1007/978-3-540-68914-0_10.

[DPP94] Ivan Damgård, Torben P. Pedersen, and Birgit Pfitzmann. On the existence of statistically hiding bit commitment schemes and fail-stop signatures. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 250–265. Springer, Berlin, Heidelberg, August 1994. doi:10.1007/3-540-48329-2_22.

[DQW23] Yevgeniy Dodis, Willy Quach, and Daniel Wichs. Speak much, remember little: Cryptography in the bounded storage model, revisited. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part I*, volume 14004 of *LNCS*, pages 86–116. Springer, Cham, April 2023. doi:10.1007/978-3-031-30545-0_4.

[DR02] Yan Zong Ding and Michael O. Rabin. Hyper-encryption and everlasting security. In *Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science*, STACS '02, page 1–26, Berlin, Heidelberg, 2002. Springer-Verlag. doi:10.1007/3-540-45841-7_1.

[DY21] Yevgeniy Dodis and Kevin Yeo. Doubly-affine extractors, and their applications. In Stefano Tessaro, editor, *ITC 2021*, volume 199 of *LIPIcs*, pages 13:1–13:23. Schloss Dagstuhl, July 2021. doi:10.4230/LIPIcs.ITC.2021.13.

[Dzi06] Stefan Dziembowski. On forward-secure storage (extended abstract). In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 251–270. Springer, Berlin, Heidelberg, August 2006. doi:10.1007/11818175_15.

[Ell02] Chip Elliott. Building the quantum network. *New Journal of Physics*, 4(1):46, 2002. doi:10.1088/1367-2630/4/1/346.

[FJP14] Luca De Feo, David Jao, and Jérôme Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *J. Math. Cryptol.*, 8(3):209–247, 2014. doi:10.1515/jmc-2012-0015.

[FYLW$^+$22] Guan-Jie Fan-Yuan, Feng-Yu Lu, Shuang Wang, Zhen-Qiang Yin, De-Yong He, Wei Chen, Zheng Zhou, Ze-Hao Wang, Jun Teng, Guang-Can Guo, et al. Robust and adaptable quantum key distribution network without trusted nodes. *Optica*, 9(7):812–823, 2022. doi:10.1364/OPTICA.458937.

[GN94] Peter Gemmell and Moni Naor. Codes for interactive authentication. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 355–367. Springer, Berlin, Heidelberg, August 1994. doi:10.1007/3-540-48329-2_30.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *28th ACM STOC*, pages 212–219. ACM Press, May 1996. doi:10.1145/237814.237866.

[GWZ22] Jiaxin Guan, Daniel Wichs, and Mark Zhandry. Incompressible cryptography. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 700–730. Springer, Cham, May / June 2022. doi:10.1007/978-3-031-06944-4_24.

[GWZ23] Jiaxin Guan, Daniel Wichs, and Mark Zhandry. Multi-instance randomness extraction and security against bounded-storage mass surveillance. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023, Part III*, volume 14371 of *LNCS*, pages 93–122. Springer, Cham, November / December 2023. doi:10.1007/978-3-031-48621-0_4.

[GZ21] Jiaxin Guan and Mark Zhandry. Disappearing cryptography in the bounded storage model. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part II*, volume 13043 of *LNCS*, pages 365–396. Springer, Cham, November 2021. doi:10.1007/978-3-030-90453-1_13.

[HJR06] K. Harmon, S. Johnson, and L. Reyzin. An implementation of syndrome encoding and decoding for binary bch codes, secure sketches and fuzzy extractors, 2006. URL: https://www.cs.bu.edu/~reyzin/code/fuzzy.html.

[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, Part II*, volume 4052 of *LNCS*, pages 192–203. Springer, Berlin, Heidelberg, July 2006. doi:10.1007/11787006_17.

[HNH13] Stefan Heule, Marc Nunkesser, and Alexander Hall. Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 683–692, 2013. doi:10.1145/2452376.2452456.

[ILL89] Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Pseudo-random generation from one-way functions (extended abstracts). In *21st ACM STOC*, pages 12–24. ACM Press, May 1989. doi:10.1145/73007.73009.

[JF11] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *PQCrypto*, volume 7071 of *LNCS*, pages 19–34. Springer, 2011. doi:10.1007/978-3-642-25405-5_2.

[LBD07a] Quoc-Cuong Le, Patrick Bellot, and Akim Demaille. On the security of quantum networks: a proposal framework and its capacity. In *New Technologies, Mobility and Security*, pages 385–396. Springer, 2007. doi:10.1007/978-1-4020-6270-4_32.

[LBD07b] Quoc-Cuong Le, Patrick Bellot, and Akim Demaille. Stochastic routing in large grid-shaped quantum networks. In *2007 IEEE International Conference on Research, Innovation and Vision for the Future*, pages 166–174. IEEE, 2007. doi:10.1109/RIVF.2007.369152.

[LBD08] Quoc-Cuong Le, Patrick Bellot, and Akim Demaille. Towards the world-wide quantum network. In *International Conference on Information Security Practice and Experience*, pages 218–232. Springer, 2008. doi:10.1007/978-3-540-79104-1_16.

[Mau92] Ueli M. Maurer. A universal statistical test for random bit generators. *Journal of Cryptology*, 5(2):89–105, January 1992. doi:10.1007/BF00193563.

[MMP^{+}23] Luciano Maino, Chloe Martindale, Lorenz Panny, Giacomo Pope, and Benjamin Wesolowski. A direct key recovery attack on SIDH. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 448–471. Springer, Cham, April 2023. doi:10.1007/978-3-031-30589-4_16.

[MU10] Jörn Müller-Quade and Dominique Unruh. Long-term security and universal composability. *Journal of Cryptology*, 23(4):594–671, October 2010. doi:10.1007/s00145-010-9068-8.

[MVZJ18] Vasileios Mavroeidis, Kamer Vishi, Mateusz D Zych, and Audun Jøsang. The impact of quantum computing on present cryptography. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 9(3):405–414, 2018. doi:10.14569/IJACSA.2018.090354.

[MW24] Giulio Malavolta and Michael Walter. Robust quantum public-key encryption with applications to quantum key distribution. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 126–151. Springer, Cham, August 2024. doi:10.1007/978-3-031-68394-7_5.

[NZ96] Noam Nisan and David Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences*, 52(1):43–52, 1996. doi:10.1006/jcss.1996.0004.

[Rab05] Michael O Rabin. Provably unbreakable hyper-encryption in the limited access model. In *IEEE Information Theory Workshop on Theory and Practice in Information-Theoretic Security, 2005.*, pages 34–37. IEEE, 2005. doi:10.1109/ITWTPI.2005.1543953.

[Reg10] Oded Regev. The learning with errors problem. In *25th Annual IEEE Conference on Computational Complexity, CCC 2010*, pages 191–204, 2010. doi:10.1109/CCC.2010.26.

[Ren08] Renato Renner. Security of quantum key distribution. *International Journal of Quantum Information*, 6(01):1–127, 2008. doi:10.1142/S0219749908003256.

[RS60] Irving S. Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8:300–304, 1960. doi:10.1137/0108018.

[Sha49] Claude E Shannon. Communication theory of secrecy systems. *The Bell system technical journal*, 28(4):656–715, 1949. doi:10.1002/j.1538-7305.1949.tb00928.x.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th FOCS*, pages 124–134. IEEE Computer Society Press, November 1994. doi:10.1109/SFCS.1994.365700.

[VM24] Nilesh Vyas and Paulo Mendes. Relaxing trust assumptions on quantum key distribution networks. *arXiv preprint arXiv:2402.13136*, 2024. doi:10.48550/arXiv.2402.13136.