Authentication Mechanisms for Open Distributed Systems

by

Antonio R. Nicolosi

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

Courant Institute of Mathematical Sciences

New York University

September 2007

David Mazières _____

Victor Shoup _____

To Nelly

Abstract

While authentication within organizations is a well-understood problem, traditional solutions are often inadequate at the scale of the Internet, where the lack of a central authority, the open nature of the systems, and issues such as privacy and anonymity create new challenges. For example, users typically establish dozens of web accounts with independently administered services under a single password, which increases the likelihood of exposure of their credentials; users wish to receive email from anyone who is not a spammer, but the openness of the email infrastructure makes it hard to authenticate legitimate senders; users may have a rightful expectation of privacy when viewing widely-accessed protected resources such as premium website content, yet they are commonly required to present identifying login credentials, which permits tracking of their access patterns.

This dissertation describes enhanced authentication mechanisms to tackle the challenges of each of the above settings. Specifically, the dissertation develops: 1) a remote authentication architecture that lets users recover easily in case of password compromise; 2) a social network-based email system in which users can authenticate themselves as trusted senders without disclosing all their social contacts; and 3) a group access-control scheme where requests can be monitored while affording a degree of anonymity to the group member

performing the request.

The proposed constructions combine system designs and novel cryptographic techniques to address their respective security and privacy requirements both effectively and efficiently.

Contents

Dedication iii

Abstract iv

List of Figures viii

1 Introduction 1

2 Coping with Password Compromise in Web Authentication 4

2.1 Introduction 5

2.2 Related Work 7

2.3 The 2Schnorr Signing Protocol 14

2.4 Implementation 35

2.5 Performance 40

2.6 Summary 43

3 E-Mail Authentication via Social Networks 45

3.1 Introduction 45

3.2 Motivating application: Re: 47

3.3 Model 50

3.4 Constructions 52

3.5 Discussion 58
3.6 Summary 61

4 Authentication and Privacy for Group Access Control 62

4.1 Introduction 63
4.2 Preliminaries 67
4.3 *Ad Hoc* Anonymous Identification Schemes 71
4.4 Generic Construction 83
4.5 Efficient Implementation 91
4.6 Applications 95

Bibliography 100

List of Figures

2.1 The 2Schnorr signature protocol. Computations carried out by the client are reported on the left; server-side computations appear to the right. m is the message being signed; the final signature is $\langle r, s \rangle$. 19

2.2 A two-message signature protocol, for applications with a bounded number of concurrent signature requests from the client. The constants are computed as in Figure 2.1. 27

2.3 SFS user-authentication architecture. 35

2.4 Messages exchanged during the user-authentication process. The authentication protocol between the agent and authd is opaque to the core file system software. 37

2.5 Implementation of proactive signatures in SFS 37

2.6 Benchmarks for signing and verifying in the Rabin and Schnorr signature schemes. End-to-end protocol shows user wait time for complete authentication. 40

3.1 A fragment of a social network. Solid arrows represent trust relationships; the dotted arrow highlights a pair of users for which to verify social proximity. 51

3.2 Data structures used for a hash-based proximity check between a sender S and recipient R. 53

4.1 Oracles for the soundness attack game. DB denotes a database storing user secret key/public key pairs, indexed by public key. . 75

4.2 The oracle for the anonymity attack game. 77

Chapter 1

Introduction

In its basic form, authentication is a well-understood concept in information security. Yet, many scenarios call for slight variations on the basic theme, where existing solutions do not directly apply; new techniques need to be developed.

In the context of password-based user authentication, for example, users often reuse the same credentials (*i.e.*, their passwords) when establishing accounts with dozens of independently administered services. Under such circumstances, a user whose password is compromised is unlikely to remember every place at which she needs to update her login information. At best, recovery from compromise is a lengthy, manual process.

Based on work first published as [70], Chapter 2 describes an authentication mechanism that addresses these challenges for SFS, a secure, global file system. To attain its goals, the system employs *proactive two-party signatures*, a special kind of digital signatures, in which a private key is split between two parties, both of whom must approve and participate in signing authentication requests. This property enables a design in which an authentication server keeps a signature log describing all network accesses performed on behalf of the user,

which provides a valuable audit trail in case of a break-in. Moreover, proactive two-party signatures allow private key shares to be updated, so that old shares cannot be combined with new ones to sign messages or to recover the private key.

While a number of proactive protocols have been proposed in the cryptographic literature, they were all based on threshold schemes that cannot be applied to the practically relevant two-party case. Our novel construction fills this deficiency, providing a solution that is at the same time easy-to-implement and cryptographically secure.

As another example, in spam-filtering systems, a legitimate sender wants to authenticate himself to the recipient as a non-spammer. Leveraging an existing social network, this type of authentication can be accomplished by demonstrating a short chain of social contacts connecting the sender to the recipient—assuming that users do not maintain social relationships with spammers.

Discovering these chains requires sharing social information, which introduces privacy concerns. Chapter 3 defines a privacy model for demonstrating proximity in social networks [47]. Using insights from this modeling, we derive efficient cryptographic protocols enabling parties to determine shared friends while exposing minimal information about their social contacts. We then describe how to integrate these privacy mechanisms within an improved prototype of the Re: (Reliable Email) white-listing system [50]. Compared with Re:'s initial design, the cryptographic solution derived from this new privacy model better addresses the system's privacy goals, and avoids the computational bottleneck due to Re:'s earlier use of a general-purpose privacy-preserving protocol.

A different flavor of authentication is at play in group access control: when

accessing a group-protected object or service, a user only needs to authenticate herself as a member of the relevant group. Controlled access to the resource is thus enforced, while affording a degree of protection to the identity of the user who "anonymously identifies" herself.

Chapter 4 introduces a cryptographic construction that allows participants from a given user population to form *ad hoc* groups [39], and then prove membership anonymously in such groups. Notably, this group-authentication protocol takes time independent of the size of the *ad hoc* group, and gives rise to the first constant-size, signer-ambiguous ring signature schemes. Key for the realization of these new multi-user protocols was our development of a cryptographic primitive, *accumulator with one-way domain*, on the line of previous work on collision-resistant accumulators.

Chapter 2

Coping with Password Compromise in Web Authentication

This chapter investigates proactive two-party signature schemes (P2SS) in the context of user authentication. P2SS allows two parties—the client and the server—jointly to produce signatures and periodically to refresh their sharing of the secret key. The signature generation remains secure as long as both parties are not simultaneously compromised between successive refreshes. We construct the first such proactive scheme based on the discrete log assumption by efficiently transforming the popular Schnorr's signature scheme into a P2SS. We also extend our technique to the signature scheme of Guillou and Quisquater (GQ), providing two practical and efficient P2SSs that can be proven secure in the random oracle model under standard discrete log or RSA assumptions.

We demonstrate the usefulness of P2SS (as well as our specific constructions) with a new user authentication mechanism for the Self-certifying File System

(SFS) [68]. Based on a new P2SS signature protocol we call 2Schnorr, the new SFS authentication mechanism lets users register the same public key in many different administrative realms, yet still recover easily if their passwords are compromised. Moreover, an audit trail kept by a secure authentication server tells users exactly what file servers an attacker may have accessed—including even accounts the user may have forgotten about.

2.1 Introduction

In an ordinary two-party signature scheme, a private key is split between two parties, both of whom must approve and participate in the signing of messages. An attacker must compromise both parties to forge signatures on its own. However, the attacker has the entire lifetime of the public key to compromise each of the two parties. Moreover, particularly in the two-party case, the parties' roles may be asymmetric—for instance, a client may have the right to initiate signatures of arbitrary messages, while a server's role is simply to approve and log what has been signed. In such settings, an attacker may well gain fruitful advantage from the use of even a single key share, unless some separate mechanism is used for mutual authentication of the two parties. Finally, ordinary two-party signatures offer no way to transfer ownership of a key share from one party to another—as the old owner could neglect to erase the share it should no longer be storing.

Proactive digital signatures allow private key shares to be updated or "refreshed" in such a way that old key shares cannot be combined with new shares to sign messages or recover the private key. While a number of proactive signature protocols have been constructed, most existing protocols are threshold

schemes designed for a variable number of parties. Because these threshold schemes require a majority of participants to be honest, they do not scale down to only two parties.

This chapter describes 2Schnorr, a proactive signature protocol specifically designed for two parties. 2Schnorr is an efficient protocol that is easy to implement and produces digital signatures compatible with the Schnorr [75] signature scheme. In the random-oracle model, a three-message version of 2Schnorr is provably secure against existential forgeries assuming only that discrete logs are hard. For applications with bounded concurrency, such as user authentication, a two-message version can also be proven secure under the stronger one-more-discrete-log assumption. The technique we describe can equally well be applied to the Guillou-Quisquater (GQ) [56] signature scheme to produce two- and three-message 2GQ protocols based on the strong RSA and one-more-RSA inversion problems, respectively. To avoid redundancy in the treatment, though, this chapter concentrates only on Schnorr signatures.

proactive two-party signature schemes (P2SS) have a natural application to the problem of user authentication, particularly in settings with many administrative realms. Within a large university, for example, it is not uncommon for a user to have five or six different shell accounts on machines in separate research groups. On the web, users typically establish accounts at dozens of different sites over time. Under such circumstances, a user whose private key or other credentials get compromised is unlikely to remember every place at which he needs to update his login information. Some of the sites may even be unavailable at the time the user tries to update them, at which point the user may just give up on the problem until the next time he needs one of the accounts.

Using 2Schnorr, we built a user-authentication mechanism that addresses these challenges for SFS [68]. SFS is a secure, global file system in which users gain transparent access to files from many different administrative realms after logging in with a single password. With the new authentication mechanism, every user has an ordinary Schnorr public signature key on file wherever the user has an account. The corresponding private key is split between the user and an authentication server of the user's choice. If the user's password is ever compromised, he can immediately block further unauthorized access to all of his accounts by updating his password and private key halves on this single authentication server. Moreover, from the server's logs, the user can determine exactly what servers an attacker has accessed, where on the network those accesses came from, and whether the attacker has changed the user's login information at any sites. Thus, even accounts the user may have forgotten about will be brought back to his attention if there is any risk of an attacker having accessed them.

The next section describes SFS and related work in user authentication and proactive signature schemes. Section 2.3 describes the 2Schnorr protocol and gives a proof of security. Section 2.4 describes the implementation of our user-authentication mechanism in SFS. Section 2.5 reports on the performance of 2Schnorr and our user-authentication scheme. Section 2.6 summarizes the results in the chapter.

2.2 Related Work

A vast number of systems have dealt with the problem of user authentication. This section describes SFS and the motivation for a new SFS user authentication

mechanism. We then highlight a few other systems that have tackled user authentication on a large scale. Finally, we discuss related work in cryptography.

2.2.1 SFS Overview

SFS is a secure network file system designed for decentralized control and easy sharing of files across organizational boundaries. In SFS's administrative model, servers are grouped into administrative *realms* that recognize the same set of authorized users. Realms can be as large as an entire campus or as small as a single server behind a DSL line. While a simple mechanism allows one realm to "import" or recognize users from another, realms in general need not trust each other, coordinate with each other, or even know of each other's existence.

Each SFS user may have accounts in many different administrative realms. From a single client machine, users can simultaneously access servers in multiple realms. The SFS client itself has no notion of belonging to a particular realm. (In fact, SFS has no client-side configuration options that would differentiate one client from another.) Users simply access files based on whatever realms they belong to. If a user accesses a file on a server the client has never heard of, an "automounting" mechanism causes the file to spring into existence before the access completes.

SFS users have public signature keys which they register with any realms in which they have accounts. User authentication consists of digitally signing an authentication request with the corresponding private key. Each user runs a program, *sfsagent*, that attempts to authenticate her to every file server she accesses. In this way, by registering the same public key in every administrative realm, a user can transparently access files from multiple realms without wor-

rying about administrative boundaries. Unfortunately, if a user's private key is ever compromised, the user may have to update her public key in a large number of realms. The mechanism described in this paper makes it considerably more difficult to compromise a user's key.

SFS comes bundled with a remote execution utility, *rex*, with similar functionality to the popular *ssh* [81]. Between the file system and *rex*, any SFS user authentication mechanism can cover a large fraction of the day-to-day network accesses people make to their servers.

2.2.2 User Authentication

Of widely used network file systems, SFS's goals are probably most similar to those of AFS [58]. AFS is a file system designed to work over the wide-area network. AFS has been particularly successful in large organizations—for instance permitting the user community of an entire university to share access to the same file systems. Unfortunately, AFS does not adapt as well to settings with many different administrative realms. AFS's security is based on the centralized Kerberos [76] authentication system in which a central authority manages all of the accounts and servers in a given administrative realm. Cross-realm authentication is possible, but requires cooperation from realm administrators. Thus, users must typically type a separate password for each realm in which they wish to access servers. Since the central Kerberos server stores a secret that is effectively equivalent to the user's password, it is inadvisable for users to have the same password in different Kerberos realms.

The SSH remote login tool supports a mode of authentication based on public keys. The user registers his private key with an agent process on the local ma-

chine, and stores the corresponding public key in a file `.ssh/authorized_keys` in his home directory on the server. SSH public key authentication is very convenient. Users therefore typically end up copying their `authorized_keys` file to all of their different accounts. Unfortunately, changing public keys requires many accounts to be updated, and users are likely to forget to update accounts on infrequently used machines.

Perhaps most relevant to P2SS are the various token- and hardware-based user-authentication systems. As smart cards and other physical security devices gain more computational power, it will become increasingly practical for them to compute digital signatures. Such configurations will be even more desirable if they can keep an audit trail of all signed messages in case the device is stolen or otherwise compromised. P2SS schemes enable such scenarios, while additionally allowing users to recover from compromised devices without changing their public keys. To compromise a user's public key permanently, an attacker would need to break the user's hardware device (or steal a backup of the user's share) *and* compromise the centralized signature server *before* the user had an opportunity to recover from the first event.

2.2.3 Two-Party Signature Generation

Ordinary two-party signature schemes are in some sense trivial. One can always take two copies of a secure one-party signature scheme (call them Sig_1 and Sig_2), publish the public keys pk_1, pk_2 for both of them, and let the first party store sk_1 and the second sk_2. A two-party signature of a message m then consists of two independent signatures of m using Sig_1 and Sig_2. The first signature can only be be produced by the first party, and the second signature by the second

party.

Most previous work on two-party signatures has therefore focused on the problem of generating signatures that are compatible with existing one-party algorithms. Such two-party schemes allow systems to interoperate with verifiers that cannot be updated to understand new signature types. While 2Schnorr and 2GQ are the first two-party schemes compatible with Schnorr and GQ, they are hardly the first schemes to interoperate with standard one-party algorithms. Bellare and Sandhu [11] and MacKenzie and Reiter [46] consider several flavors of two-party generation of the RSA (full domain hash) signature scheme (building on some previous less formal work, *e.g.*, [20, 49]). The schemes are simple, elegant, and in most cases reducible to the basic RSA assumption. MacKenzie and Reiter [67] also give a protocol for two-party generation of DSA signatures [45].

More closely related work was proposed in [65], where MacKenzie and Reiter extend their schemes from [66] to allow for *delegation* of password-checking services. As noted by the authors, this extra property offers an approach to proactively update the password-protected secret key of a networked device.

The resulting P2SS is designed and optimized for a hardware-based user-authentication model, whereas the primary motivation of our study of P2SS is to obtain general-purpose schemes that could be combined with any user authentication mechanism.

Indeed, in the model of [65], password authentication and signature generation are tightly combined together, and performed by a password-protected, networked device which stores sensitive data on behalf of the user, and (encrypted) tokens for the server. Hence, to authenticate to the server and obtain a signature, the user needs possession of a physical device.

In contrast, in our model mutual authentication between the two parties involved in the P2SS is separated from the actual joint signing protocol, coherently with the spirit of modularity of the design of SFS [68]. No matter how the two parties establish a secure, authenticated channel, 2Schnorr and 2GQ let them jointly generate signatures in a proactive secure manner. As mentioned in Section 2.4, this can be combined with password-based authentication protocols, allowing a user to access her cryptographic services from any machine around the world, with no need of bringing along any physical device, as long as she remembers her password and trusts the local software to erase all sensitive information at the end of her session.

Our approach thus allows for a wider class of usage scenarios, since people do not always carry smartcards or PDA's—whereas our new user authentication mechanism has been implemented completely in software, and is already used in the daily work of the authors. On the other hand, the customized P2SS resulting from [65] may be more desirable for the specific setting thy consider, and we could indeed adapt our 2Schnorr and 2GQ schemes to their model to get similar protocols based on different assumptions.

Two-party signatures can also be viewed as a special case of general secure two-party computation [80] and threshold cryptography [37]. However, most threshold cryptography results assume an honest majority and therefore apply only to $n \geq 3$ players. Many threshold techniques and even definitions (*e.g.*, robustness) are inapplicable to the two-party setting.

Two-party signatures are also related to the notion of *key-insulated signature scheme* [38]. In this model, there is a server that helps the client update its secret key from period to period (while keeping the same public key). Within a period, however, the client performs all signatures on its own. This has the

advantage of not requiring the server's cooperation on each signature, but for our application we specifically *want* all signatures to go through the server for approval and logging. (This is also what allows very simple key revocation when the client's key is compromised.)

2.2.4 Proactive Security

A basic two-party signature scheme remains secure only as long as both parties are not compromised. Unfortunately, the longer the lifetime of the public key, the more realistic the concern that *both* the client and the server may at one point have been compromised. General *proactive cryptosystems* [71, 57] address this problem by allowing potentially unbounded number of compromises, as long as not too many happen simultaneously. Specifically, there is an efficient share *update* protocol which allows players to refresh their current sharing of the secret key. As long as not too many servers are compromised between any two successive refreshes, the system remains secure.

As with threshold cryptography, proactive cryptography has concentrated on $n \geq 3$ players. To the best of our knowledge, proactive signature schemes have not previously been studied in the two-party setting, except for the brief remark in the aforementioned work of [65].

Recent work of Itkis and Reyzin [60] on intrusion-resilient signatures describes a setting similar to P2SS. These combine the properties of key-insulated and proactive signatures. However, as in the key-insulated model, the server only helps the client to update its secret key from one time-period to another; all the signing is done by the client alone. In the P2SS model, the actual secret does not change from one time period to the next; only the sharing of the secret

changes.

In both 2Schnorr and 2GQ, the share update protocol is very simple: a client simply sends a random element of an appropriate group to the server over a secure channel. Of course, this does not mean that proactivization is generally simple in the two-party case. Indeed, there seems to be no way to "proactivize" the generic double signature two-party approach, so the question of *generic* proactive two-party signature schemes is not as trivial as in the non-proactive case. Except for the two-party RSA scheme of [66], where proactivization comes at the cost of some efficiency loss (due to the need of resorting to the techniques of [46] to share the secret over a much larger modulus), previous two-party signatures do not appear to "proactivize" in as simple way as our 2Schnorr and 2GQ schemes do.

2.3 The 2Schnorr Signing Protocol

This section specifies the 2Schnorr signing protocol and analyzes its security. Like the standard Schnorr signature scheme, 2Schnorr relies on a cryptographic hash function, H, which for the proofs we will assume behaves like a random oracle—an assumption very common in the cryptographic research, first formalized in [9]. Before going into the details of 2Schnorr, we briefly describe the standard Schnorr scheme.

The Schnorr signature scheme was first proposed in [74] as an application of the Fiat-Shamir transformation [43], and its security has been analyzed, among the others, in [75, 72]. It can be instantiated on every group $\mathbf{G}$ of prime order where the discrete log problem is believed to be hard, and can be proven secure under the standard notion of *existential unforgeability against the*

adaptive chosen-message attack [55] in the Random Oracle Model, assuming that computing discrete logs in the underlying group is hard. For concreteness, we will consider cyclic subgroups of $\mathbf{Z}_p^*$ (for large primes p) of prime order q.

The key generation algorithm produces two large prime p and q such that $q|(p-1)$, and an element g of $\mathbf{Z}_p^*$ of order q. Then it picks a random element x in $\mathbf{Z}_q^*$, and sets $y = g^x \bmod p$. The public key is now $\langle p, q, g, y \rangle$, while the corresponding private key is x. Notice that the group parameters p, q, g can be safely shared between a community of users, so that y by itself can be thought as the public key corresponding to the private key x. We will also assume that a cryptographic hash function H mapping arbitrary strings into elements of $\mathbf{Z}_q^*$ has been specified as a parameter of the scheme.

To sign a message m, the holder of the private key x picks a random $k \in \mathbf{Z}_q^*$ and set $r = g^k \bmod p$. It then computes $e = H(m, r)$, $s = k + xe \bmod q$, and outputs the signature $\langle r, s \rangle$. Notice that k must be kept secret and chosen anew each time: disclosing or reusing the value of k would allow recovery of the secret key x.

To check whether a given $\langle r, s \rangle$ is indeed a signature for some message m, it suffices to know the corresponding public key $\langle p, q, g, y \rangle$ and verify that $g^s = ry^e \bmod p$, where $e = H(m, r)$.

The Schnorr signature scheme is often studied together with the GQ scheme, its "twin" based on the RSA assumption. In fact, they can be thought as variations of the same basic theme. Semantically, the difference between the two schemes is that in Schnorr's all secrets are drawn from the additive group $(\mathbf{Z}_q, +)$ and their public counterparts are obtained by exponentiation on a fixed base; in the GQ scheme, instead, private data is taken from the multiplicative group $(\mathbf{Z}_N^*, *)$ (where N is the product of two big primes) and public quantities are

obtained by exponentiating to a fixed exponent. Syntactically, this is equivalent to convert all additions in Schnorr's scheme into multiplications, and all multiplications (involving secrets) into exponentiations. A mechanical application of such "conversion rules" to our 2Schnorr protocol (described below) yields the 2GQ protocol, which enjoys analogous security properties based on the RSA assumption.

2Schnorr is a simple and elegant two-party proactive variation of the above scheme. We call the two parties the *client* and the *server*. We assume the client is the party that wants the digital signature—it starts with the message and ends with the signature. The server simply wants to log or approve all signed messages.

The main issue in obtaining a two-party solution for Schnorr signatures is that if one party (say the client) could control the choice of one of the secret quantities x or k, or their public counterparts y and r, then the client would gain an advantage over the server, and the resulting scheme might not be secure.

Fortunately, in our case the parties need just to agree on a random value, a task usually referred to as *coin flipping* [14]. This can easily be achieved by having each party choosing its random share, and then exchanging and combining the two shares. To avoid any unfairness in the exchange due to the fact that one party (say the server) must reveal its share first, the server will only "commit" to its random share, and will "open" the commitment only upon receiving the client's random value.

Since the value the server has to commit to is a random share, the commitment can be achieved using a cryptographic hash function G that behaves as a random oracle: to commit, the server will just send the hash of its random share; the client will not be able to learn anything from it, but will be able to

check the consistency of the server's randomness at the end of the exchange. Notice that although we already introduced a random oracle H, we are using a different notation for G as a syntactic reminder that the two oracles H and G serves different purposes, and that one of them (*i.e.*, G) could be replaced by any other "extractable" commitment scheme (like "committing" encryption; *cf.* [34]).[1] Furthermore, G and H take inputs of different length. In an actual application, both oracles could be conveniently implemented in term of a cryptographic hash function like SHA-1 [44], prepending a 0 for H and a 1 for G:

$$H(z) = \text{SHA-1}(0, z) \qquad G(z) = \text{SHA-1}(1, z).$$

Remark. Our use of extractable commitment to solve the problem of the randomness in Schnorr and GQ schemes generalizes to other probabilistic signature schemes where the randomness is public, but its choice plays a crucial role for the claimed security result (*e.g.*, PSS [10], PFDH [31]).

Key generation. A 2Schnorr public key is just an ordinary Schnorr public key, $\langle p, q, g, y\rangle$. However, the corresponding Schnorr private key, x, is split between two key halves, x_c and x_s, such that $x \equiv x_c + x_s \pmod{q}$. A public 2Schnorr key can be centrally generated along with two halves for the corresponding private key as follows:

$$\begin{aligned} q &\leftarrow \text{a large prime} \\ p &\leftarrow \text{a larger prime such that } q|(p-1) \\ g &\leftarrow \text{an element of } \mathbf{Z}_p^* \text{ of order } q \\ x_c, x_s &\leftarrow \text{random elements of } \mathbf{Z}_q \\ y &= g^{(x_c+x_s)} \bmod p \end{aligned}$$

The two private keys halves are x_c and x_s.

[1] Extractable commitments can be realized without the random oracle model, but we are using random oracles anyway, so we go for the simpler design.

For distributed key generation between the client and the server, the client chooses p, q, g, and x_c, computes $y_c = g^{x_c} \bmod p$, and sends the server $\{p, q, g, G(y_c)\}$. The server then picks x_s and sends the client $y_s = g^{x_s} \bmod p$. Finally, the client reveals y_c, both parties compute $y = y_c y_s \bmod p$, and the public key is $\langle p, q, g, y \rangle$. To prevent the client from (maliciously) choosing "bad" group parameters [13], the server may require from the client a certification that the values p, q, g have been generated according to some specific algorithm: in the implementation described in Section 2.4, we actually used the method proposed by the NIST in [45], for which such a proof can be easily provided.

Signature generation. To sign a message m, the client and the server each select a random element of $\mathbf{Z}_q$—k_c for the client, k_s for the server. The two parties then exchange the three messages shown in Figure 2.1. First the server picks at random an ephemeral private key k_s from $\mathbf{Z}_q^*$, computes the corresponding ephemeral public key $r_s = g^{k_s} \bmod p$ and sends the G-hash of it (message **1**). Similarly, the client computes its ephemeral key pair $\langle k_c, r_c \rangle$ and sends the second flow of the protocol, consisting of the value $G(r_s)$ it got in message **1**, its ephemeral public key r_c and the message m it wishes to sign. Upon receiving message **2**, the server checks that r_c belongs to the group specified by p, q and g by verifying the equality $r_c^q \bmod p \stackrel{?}{=} 1$, then computes $r = r_c r_s \bmod p$, $e = H(m, r)$, $s_s = k_s + x_s e \bmod q$, and replies with message **3**, which reveals the value of r_s. The client computes $G(r_s)$ and verifies that it matches the hash value received in message **1**; if so, it verifies r_s and computes s_c in a way analogous to what described above for the server. Finally, the client sets $s = s_c + s_s \bmod q$, and obtain the pair $\langle r, s \rangle$—an ordinary Schnorr signature.

The protocol in Figure 2.1 requires three messages. The first message can be precomputed and sent to the client in advance, reducing network latency to

$$
\begin{aligned}
k_c &\xleftarrow{R} \mathbf{Z}_q & k_s &\xleftarrow{R} \mathbf{Z}_q \\
r_c &= g^{k_c} \bmod p & r_s &= g^{k_s} \bmod p \\
r &= r_c r_s \bmod p & e &= H(m, r) \\
s_c &= k_c + x_c e \bmod q & s_s &= k_s + x_s e \bmod q \\
s &= s_c + s_s \bmod q
\end{aligned}
$$

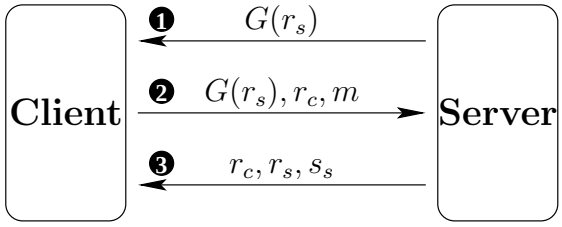

Figure 2.1: The 2Schnorr signature protocol. Computations carried out by the client are reported on the left; server-side computations appear to the right. m is the message being signed; the final signature is $\langle r, s \rangle$.

a single round trip from the time the client receives the message to be signed. As discussed later, however, a system with a constant bound on the number of concurrent signatures requested by the client can simply eliminate the first message of the protocol (and remove $G(r_s)$ from the second message). The resulting two-message protocol may be more convenient to implement.

Signature verification. Signature verification is identical to Schnorr. Given a public key $\langle p, q, g, y \rangle$, a message m, and a signature $\langle r, s \rangle$, the signature is valid if and only if $g^s \equiv r y^e \pmod{p}$, where $e = H(m, r)$. It can be easily checked that signatures obtained from an honest execution of the signing protocol do indeed verify correctly.

Key update. The aim of key updates is to tolerate multiple break-ins of each party. Clearly, if an adversary A could break in both parties between two

successive key updates, A will learn the entire state of the system, so that no security can be guaranteed any more. Still, the system should be able to withstand multiple break-ins as long as one *honest key update* happens in between.

Key updates are considerably simpler in the two-party case than in general proactive signatures. Since either party can already destroy the private key by erasing its own share, there is no need to preserve the private key when the client or server misbehaves. Further simplifying the problem, our update protocol assumes a secure channel between the client and server, because our system already requires such a channel for other purposes. (Otherwise, the client and server could use a 2Schnorr signature as part of negotiating a secure channel.)

To update the key halves x_c and x_s, the client picks a random $\delta \in \mathbf{Z}_q$ and sends it to the server over a secure channel. The new key halves x'_c and x'_s are simply computed as: $x'_c = x_c - \delta \bmod q \qquad x'_s = x_s + \delta \bmod q$

Such key update is not only quite simple, but also very effective in re-randomizing the state of the system, thus simplifying the arguments for the security proofs in the following.

2.3.1 Security against Malicious Clients

Theorem 1. *If a malicious client can forge a signature without the approval of the server in probabilistic polynomial time (PPT) with non-negligible probability, then we can also compute discrete logs in PPT with non-negligible probability.*

Proof. Let p, q, and g be as in Schnorr public keys. Let y, in the group generated by g, be a random element of which we wish to compute the discrete log. Let A be an adversary that behaves like a 2Schnorr client but then outputs a

forged signature (which the server did not approve) in PPT (with non-negligible probability).

Choose a random element $x_c \in \mathbf{Z}_q$. Let $y_c = g^{x_c} \bmod p$ and $y_s = y y_c^{-1} \bmod p$. Give A public key $\langle p, q, g, y \rangle$ and private key x_c. Now ask A to forge a signature.

A can make four types of oracle query we must respond to. It can make random oracle queries to the hash functions H and G. It can make update queries to refresh the key halves. It can ask the server to start a signature (corresponding to message **1** in the protocol). Finally, it can ask the server to endorse a signature (*i.e.*, return the third flow in the protocol).

Without loss of generality, we will assume throughout this paper that A does not ask for the same hash query to the same oracle twice (A could simply store previously computed values in a table). All oracle queries to G are answered with random values. As for oracle queries to H, we reply to them with random but consistent answers: however, during certain other queries (described below), we set the value of the random oracle H on certain inputs to specific, though uniformly distributed, values.

We keep a running total, Δ, of all the update requests A makes. Initially, $\Delta = 0$, but for each update request δ, we add δ to $\Delta \pmod q$.

When A makes a start signature query, we choose two random numbers $\alpha, \beta \in \mathbf{Z}_q$ and compute $r_s = g^\alpha y_s^\beta \bmod p$. We now fix some random output for the value of $G(r_s)$, and return this to A. Notice that A cannot learn anything about r_s from the random value $G(r_s)$, unless it has previously asked for the hash of r_s to the oracle G; however, the chance of A having already asked for $G(r_s)$ is negligible, since r_s is uniformly distributed in the group of order q generated by g.

When A asks us to endorse a signature with query $\{G(r_s), r_c, m\}$, we com-

pute $r = r_c r_s \bmod p$, and fix the random oracle value of $e = H(m, r)$ such that $H(m, r) \equiv -\beta \pmod q$. Notice that the probability of having already set this value due to a previous hash query to H is negligible, since up to this point r_s is a random element unknown to A (A just saw $G(r_s)$, a random value that leaks no information about r_s) and so r is also uniformly distributed in the group generated by g. We set $s_s = \alpha + \Delta e \bmod q$, and return $\{r_s, s_s\}$. If A was talking to an ordinary server, the server would reply with $s_s = k_s + (x_s + \Delta)e \bmod q$, where k_s is the discrete log of r_s and x_s is the discrete log of y_s. Even though we cannot compute k_s and x_s, we are still computing the correct value of s_s by returning $\alpha + \Delta e$:

$$\begin{aligned}
g^{\alpha} y_s^{\beta} &\equiv r_s \pmod p \\
g^{\alpha} &\equiv r_s y_s^{-\beta} \pmod p \\
g^{\alpha} &\equiv g^{k_s} (g^{x_s})^{-\beta} \pmod p \\
g^{\alpha} &\equiv g^{k_s} g^{x_s e} \pmod p \\
\alpha &\equiv k_s + x_s e \pmod q \\
\alpha + \Delta e &\equiv k_s + (x_s + \Delta)e \pmod q
\end{aligned}$$

Because our responses to A's oracle queries are indistinguishable from those of a real server and random oracles, A will output with non-negligible probability some message and forged signature $m, \langle r, s \rangle$.

We can now rewind A's state to the first time it queried the random oracle for $e = H(m, r)$, and return some new, randomly chosen value $e' \neq e$. Such rewinding argument is quite common in analyzing the security of similar schemes [59, 8, 7]: intuitively, since A only makes polynomially many oracle queries and e' is statistically indistinguishable from e, there is a non-negligible

probability that A will again forge a signature for the same m and r, yielding a second signature $\langle r, s' \rangle$ for m, with $s' \neq s$. The exact probabilistic analysis is based on the *forking lemma* of [72], and is quite similar to the one for the standard Schnorr signature scheme.

By the verification property, $\langle r, s, e \rangle$ and $\langle r, s', e' \rangle$ satisfy the following two congruences:

$$\begin{aligned} g^s &\equiv r g^{xe} \pmod{p} \\ g^{s'} &\equiv r g^{xe'} \pmod{p} \end{aligned}$$

By memberwise division of the above congruences, we can compute x, the discrete log of y, as follows:

$$\begin{aligned} g^{s-s'} &\equiv g^{x(e-e')} \pmod{p} \\ s - s' &\equiv x(e - e') \pmod{q} \\ x &= (s - s')(e - e')^{-1} \bmod q \end{aligned}$$ □

2.3.2 Security against Malicious Servers

Theorem 2. *If a malicious server can forge a signature without the client's help in PPT with non-negligible probability, then we can also compute discrete logs in PPT with non-negligible probability.*

Proof. Let p, q, g, y be a challenge in which we need to find the discrete log of y. Let A be an adversary that acts as a 2Schnorr server and can forge signatures. We choose a random $x_s \in \mathbf{Z}_q$ and compute $y_s = g^{x_s} \bmod p$ and $y_c = y y_s^{-1} \bmod p$. Give A public key $\langle p, q, g, y \rangle$ and private key x_s, and ask it to forge a signature.

As before, A can make four types of oracle query: random oracle queries (to H or to G), update queries, asking the client to initiate the signature of a

particular message, and asking the client to finish computing the signature of a message for which an initiate query was previously done.

The two random oracles are treated as before. For update queries, we are now even allowed to choose a random δ ourselves: since the client refreshes its key half by subtracting δ, here Δ is defined as the running total of the values $q - \delta$ for each update (mod q).

When A asks to initiate the protocol, we choose two random numbers $\alpha, \beta \in \mathbf{Z}_q$ and compute $r_c = g^\alpha y_c^\beta \bmod p$. Notice that we allow A to choose the message m it wants to sign, but still, A must provide its "commitment" $G(r_s)$. Since G behaves like a random oracle, A must have asked for the value of $G(r_s)$—otherwise it will have only a negligible chance of guessing the correct value needed to open, in the third flow of the protocol, the commitment sent in the first message. Hence, upon receiving an initiate query, we can perform a simple look up for $G(r_s)$ in the table containing the pairs $\langle$*query, answer*$\rangle$ for all the oracle queries that A has done to G so far.[2] Then, we compute $r = r_c r_s \bmod p$ and fix the random oracle value of $e = H(m, r)$ such that $H(m, r) \equiv -\beta \pmod q$: e is still uniformly distributed in Z_q since we chose β at random.

The complete signature query are handled virtually identically to the proof of Theorem 1, reversing the s and c subscripts in variables. The only difference is that when completing a signature, we must return $\langle r, s\rangle$ instead of $\langle r_c, s_c\rangle$. r and s are easy to compute given r_c, r_s, s_c, s_s, all of which we have.

Since the interaction of A with the oracles thus simulated is indistinguishable from a real attack to the 2Schnorr signing protocol, with non-negligible probability A will forge a signature $\langle r, s\rangle$ for a message m that the client did

[2] As we mentioned, we can replace G by any "extractable" commitment which would also allow us to recover r_s in our simulation.

not finish signing. Therefore, using the same technique as in Theorem 1, we can compute the discrete log of y with non-negligible probability. □

2.3.3 Security against Mobile Adversaries

Theorem 3. *If the 2Schnorr protocol is secure against both malicious clients and malicious servers, then it is also secure against a adversary that repeatedly breaks into the client and the server, as long as a honest key update happens between any two consecutive break-ins.*

Proof. Intuitively, the reason why the proactive security of 2Schnorr follows from the security against malicious clients and against malicious servers is that once the adversary A misses a refresh of the shares x_c and x_s, the new state of the system is independent from all the information A could have learnt during its previous interaction.

More formally, assume that by repeatedly breaking into the client and the server (while always allowing a honest key update to happen in between) A is able to forge a signature with some non-negligible probability ε. Assume also that A finally obtains such forgery while in control of the client more often than half of the times (the case when A is most successful when finally breaking into the server is completely analogous). In other words, with probability $\varepsilon/2$, after several (non-simultaneous) compromises of both parties, at the end A successfully forges a signature without the server endorsing it.

We now show how to construct a malicious client A' that, using A as a black box, forges signature without the server's approval with (non-negligible) probability $\varepsilon/2$.

A' is given the public key $\mathrm{PK} = \langle p, q, g, y \rangle$, the client's share x_c and access

to a server oracle willing to engage in as many protocol runs as A' wishes. To the aim of computing a signature without the server's help, A' runs A, initially feeding it with the public key PK. For all break-ins (except for the last one) that A may want to carry out, A' follows the simulation from Theorem 1 or Theorem 2, according to whether A asked to compromise the client or the server. Notice that since we are assuming a honest key update to occur between any two consecutive break-ins, and since each time the sharing of the secret key x as $x_c + x_s$ is completely re-randomized, there is no problem with all these simulations, which will proceed exactly as A expects. Thus, at the end A will ask to carry out its final break-in against the client. This time A' gives A its own secret share x_c, and forward all A's query to the server to its own server oracle.

Overall, A' was able to let A mount its mobile attack against the system. Hence, with (non-negligible) probability $\varepsilon/2$, A will finally forge a signature that the server did not approve. A' then simply outputs such signature as its forgery, thus succeeding with the same non-negligible probability $\varepsilon/2$ in its own (malicious-client-only) attack against the system. But this contradicts our hypothesis, and the theorem follows. □

2.3.4 Two-Message Protocol

In certain cases, it may be desirable to compute signatures with only two messages. If there is a constant bound on the number of concurrent signatures a client requests, the first message can be eliminated (and the second flow consequently modified removing the hash value $G(r_s)$), yielding the simpler two-message protocol in Figure 2.2.

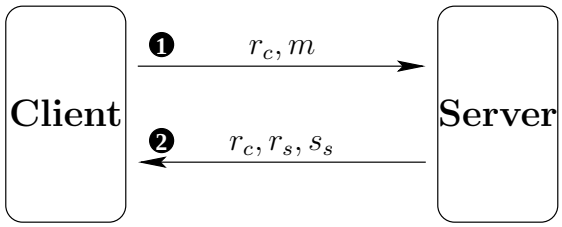

Figure 2.2: A two-message signature protocol, for applications with a bounded number of concurrent signature requests from the client. The constants are computed as in Figure 2.1.

From Theorem 1, we see that the two-message protocol is secure against a malicious client. Indeed, in this case the malicious client is just less powerful than in the previous scenario (since it does not have access to the initiate signature oracle): If an adversary A could forge signatures by acting as a two-message client, one could trivially build a three-message adversary A' in terms of A.

Furthermore, the proof of Theorem 3 is clearly not affected by the change in the signing protocol, since that proof hinges solely upon the properties of the key update protocol. Hence, the proactive security of the two-message protocol will follow as soon as its security against malicious server is established.

To prove the two-message protocol secure against a malicious server, however, we must rely on a stronger assumption than the difficulty of computing discrete logs, and assume a constant bound t on the number of concurrent runs of the protocol[3]. The intuitive reason behind the strengthening of the assumptions is that in the two-message protocol the server can control the value of r by (maliciously) choosing $r_s = rr_c^{-1} \bmod p$. In this way, the server can be able

[3]More precisely, we assume that the client is willing to initiate an unbounded number of concurrent executions of the protocol, but it will only complete one of the t most recent pending runs.

to compute $H(m, r)$ before sending the value r_s to the client. This would break the simulation described in the Section 2.3.2, so that the proof does not go through. Still, it is not clear how this attack could help the server in forging a signature $\langle r, s \rangle$, since the choice of $r_s = r r_c^{-1}$ leaves the server with the problem of computing its the discrete log, *i.e.*, k_s; hence, the server will not be able to compute the correct $s_s = k_s + (x_s + \Delta)e \bmod q$ needed to complete the signature $\langle r, s \rangle$.

Indeed, we show that the modified scheme *is secure* under the assumption that the *known-target discrete log problem* (DL-KT) [8] is hard. The DL-KT problem consists of getting some number n of discrete log challenges in the same group, then computing their discrete logs while making at most $n - 1$ queries to a discrete log oracle. Although the assumption that the DL-KT problem is hard for any polynomially-bounded n is relatively new, it has already been used quite a bit in proving the security of various schemes [8, 7, 15]. However, in all these other cases the claimed result follows almost trivially from such assumption. On the contrary, our application is considerably more involved: it requires an additional "twist" in the reduction argument, namely the assumption on the bounded concurrency, and the development of some novel ideas in the probabilistic analysis.

Theorem 4. *If a malicious server, interacting with a client with bounded concurrency, can forge a signature without the client's help in PPT with non-negligible probability, then we can also solve the DL-KT problem in PPT with non-negligible probability.*

Proof. Let p, q, g define a group as usual. Let $z_0, z_1, z_2, \ldots$ be challenges of which we want to compute the discrete logs. Let t be the concurrency bound

of the client. Let A be an adversary that acts as a 2Schnorr server and forges signatures in PPT. We will compute the discrete logs of n challenges for some number $n > 0$, while making only $n - 1$ queries to a discrete log oracle.

We first choose a random $x_s \in \mathbf{Z}_q$. Let $y = z_0$ (*i.e.*, the first challenge), $y_s = g^{x_s} \bmod p$, and $y_c = y y_s^{-1} \bmod p$. We give A public key $\langle p, q, g, y \rangle$ and private key x_s. Finally we ask A to forge a signature.

Again, A can make four kinds of oracle queries. It can query the random oracle, ask for a key update, ask the client to initiate the protocol on some message, or ask the client to complete and output a signature. We emulate the random oracle H in a completely honest manner, by replying to each query with a random value. More specifically, we choose a random (and sufficiently long) list of values for the hash oracle H *before* we even start executing A: we will use these values one after the other to answer A's queries, no matter what specific value A is asking for. Also, we randomly choose and *fix* the entire random tape of A. Finally, we also choose update values δ randomly, and keep a running sum $\Delta = \sum_\delta (q - \delta) \bmod q$. To put it differently, all the randomness we need for the entire simulation is chosen and fixed.

When A asks us to initiate the signature on a message m, we set $r_c = z_j$ for some challenge z_j we have not yet used, and we send A the message $\{r_c, m\}$.

When A wishes us to complete the signature of some m, it will send us $\{r_c = z_j, r_s, s_s\}$. Let $r = r_c r_s \bmod p$ and $e = H(m, r)$. We query our own discrete log oracle to find the discrete log of $r_c (y_c g^\Delta)^e \bmod p$. Let s_c be this logarithm. Let k_c be the log of r_c and x_c be the log of y_c. Both k_c and y_c are

unknown to us. However:

$$\begin{aligned} g^{s_c} &\equiv r_c(y_c g^{\Delta})^e \pmod{p} \\ g^{s_c} &\equiv g^{k_c}\left(g^{x_c+\Delta}\right)^e \pmod{p} \\ s_c &\equiv k_c + (x_c+\Delta)e \pmod{q} \end{aligned}$$

Thus, s_c is exactly as it should have been. We compute $s = s_c + s_s \bmod q$, and output the signature $\langle r, s\rangle$.

Since we are exactly simulating the attack scenario, A will eventually output, with non-negligible probability, a message m and forged signature $\langle r, s\rangle$.

We then rewind A's state to the time it queried the random oracle for the value of $H(m, r)$ (say this was the i^{th} A did to the oracle H). Call this query *crucial*. This time, instead of using the value in the "list of randomness" we prepared before beginning executing A, we discard this value, and answer such *crucial* query with a new, random value. We then continue the simulation as in the first execution: in particular, we will continue using our "list" to reply to random oracle queries, but will use brand new (already prepared) challenges z_j.

As we will shortly prove in Lemma 5, there is a non-negligible probability that A will forge again the same message m with randomness r but different values for $H(m, r)$ (plus some additional property will hold; see below). Once we come up with two signatures for the same message m, the same randomness r but *different* hash values $e \neq e'$ (and hence $s \neq s'$), it is just a matter of modular arithmetic to recover x (*cf.* Theorem 1). Now, given x, we can answer all the challenges z_j as follows.

For each initiate query that the adversary decided not to complete (*i.e.*, he initiated a run of the protocol but then it decided *not* to complete it), we have

used one of the challenge z_j without "consuming" any query to our discrete log oracle. Therefore, we can use it now to find the discrete log of z_j. So we concentrate on the initiate queries which were completed by A.

Next, since $y = g^x \bmod p = z_0$, x itself is the answer to the challenge z_0 (and we did not consume any discrete log queries for getting x!). Moreover, once we know both x and x_s, we can also compute $x_c = x - x_s \bmod q$. Given x_c, we can recover the value k_c—the discrete log of each z_j—from each of the $s_c = k_c + (x_c + \Delta)e \bmod q$ values we asked our discrete log oracle to compute.

There is only subtle problem: we have to ensure that we consume at most one discrete log query per each value z_j (for $j \geq 1$), *i.e.*, per each (completed) initiation query of the adversary. Such queries can be divided into three parts: the queries initiated and completed before the critical hash query i, the (at most t) queries initiated before but completed after the critical query, and the queries initiated and completed after the critical query. There is no problem with the first and the last kind of queries. Indeed, they both consumed exactly one z_j and utilized exactly one call to the discrete log oracle (recall, we use fresh z_j's in the second run). However, the second category could present problems: each query utilizes one z_j, but can potentially call the discrete log oracle *twice*, *i.e.*, once in each run of A. Here is where we will use that t is bounded (by a constant). For each such "semi-completed" signature query m_j, let r_j be the corresponding randomness in the first run, and call the t hash queries $H(m_j, r_j)$ *important* (in the first run). Notice, important queries could appear both before and after the critical query i, since the server can control the randomness by choosing a "cheating" value of $r_{s,j} r_j r_{c,j}^{-1} \bmod p$ for each m_j, and query $H(m_j, r_j)$ way before the initiation query for m_j. Similarly, we can define these t important hash queries in the second run (corresponding the the same m_j, $r_{c,j}$ but possible

different values r'_j, $r'_{s,j}$). However, since we chose all the randomness for our hash query answers at the beginning, and reused this randomness in the second run, we do not have to make the extra t calls to the discrete log oracle provided *the important queries in the first and the second run have the same indices.* Indeed, in this case we would return the same value e_j for the important query j in both runs, and therefore will need to compute the discrete log of two very similar values $r_{c,j}(y_c g^{\Delta_j})^{e_j} \bmod p$ and $r_{c,j}(y_c g^{\Delta'_j})^{e_j} \bmod p$. It is clear that these discrete logs differ by $(\Delta_j - \Delta'_j)e_j \bmod q$, which is a known value, so we can compute the discrete log $s'_{c,j}$ by returning $(s_{c,j} - (\Delta_j - \Delta'_j)e_j) \bmod q$. We will argue in Lemma 5 that (for a bounded t) the important queries will indeed be the same with non-negligible probability.

To summarize, if we are lucky that the following three conditions hold during the "double" run of A, we utilize one less discrete log query than the number of discrete log challenges we are computing, thus breaking the DL-KT assumption:

1. A succeeded the first time on some critical query i.
2. A succeeded the second time on the same critical query i, but the i^{th} response we gave was different.
3. A used the *same* (up to) t important queries in the first and second run.

We argue that the above three conditions hold (with non-negligible probability) in the following final lemma, which completes the proof. □

Lemma 5. *Let ε be the probability of A successfully producing a forgery in one run, and δ be the success probability of satisfying the above three conditions in the "double" run of A. Assume also that A makes at most q_{hash} random oracle queries in one run, and t is the maximum number of concurrent signature*

queries the client initiates. Then $\delta \geq \varepsilon^2/q_{\mathsf{hash}}^{t+1} - \mathsf{negl}(k)$. *(here* k *is the security parameter and* $\mathsf{negl}(k)$ *is a negligible function in* k*).*

Proof. Without loss of generality, we will assume throughout the proof that A always calls the random oracle H to validate all issued signatures and its final forgery—otherwise A has at most a negligible success probability, which is consumed in the $\mathsf{negl}(k)$ term. Let $i \in [q_{\mathsf{hash}}]$ be an index and $J \subset [q_{\mathsf{hash}}]$ be a subset of at most t indices. Let $\varepsilon_{i,J}$ be the probability of adversary's success in one run given that the critical query corresponding to the forgery is query number i, and that all the (at most) t important queries have indices in J. Obviously, $\sum_{i,J} \varepsilon_{i,J} = \varepsilon$.

Next, recall δ is the probability of success of the above experiment. For any i and J defined as above, let $\delta_{i,J}$ be the success probability conditioned on the fact that in both runs the critical query is i and the important queries were in the set J. Again, it is clear that $\sum_{i,J} \delta_{i,J} = \delta$.

We claim that if the size of the universe of hash responses has size $L \geq 2^\ell$, we have:

$$\delta_{i,J} \geq \varepsilon_{i,J}^2 - \varepsilon_{i,J} 2^{-\ell} \tag{2.1}$$

Assuming Equation (2.1) is true, by using Cauchy-Schwartz inequality[4] we get:

$$\begin{aligned}\delta &= \sum_{i,J} \delta_{i,J} \geq \sum_{i,J} (\varepsilon_{i,J}^2 - \varepsilon_{i,J} 2^{-\ell}) \\ &= \sum_{i,J} \varepsilon_{i,J}^2 - \varepsilon 2^{-\ell} \geq \frac{1}{q_{\mathsf{hash}}^{t+1}} \Big(\sum_{i,J} \varepsilon_{i,J} \Big)^2 - \frac{\varepsilon}{2^\ell} \\ &= \frac{\varepsilon^2}{q_{\mathsf{hash}}^{t+1}} - \frac{\varepsilon}{2^\ell} = \frac{\varepsilon^2}{q_{\mathsf{hash}}^{t+1}} - \mathsf{negl}(k)\end{aligned}$$

It remains to show the validity of Equation (2.1). For any fixed i and J, choose at random the random tape of the adversary, q_{hash} answers to the hash queries, plus any other randomness our simulator needs. Fix this randomness, but split it into the two parts, and consider the following matrix. On the columns we place the random response to query number i (call this response e); on the rows of the matrix we place all the remaining randomness (call it R). Mark a square in this matrix if the square corresponds to adversary's success in a *single* run (*i.e.*, we do not rewind it yet). Let $\varepsilon_{i,J,R}$ be the probability of success conditioned on R, *i.e.*, the density of marks in row number R of our matrix. Obviously,

$$\varepsilon_{i,J} = \frac{1}{|R|} \sum_R \varepsilon_{i,J,R} \tag{2.2}$$

Next, let $\delta_{i,J,R}$ be the probability of success of the whole experiment (with rewinding), conditioned on the fact that the row was R, *i.e.*, conditioned on the fact that the randomness used in the entire experiment (except for the answer to query number i) is described by R. We want to estimate the probability that, when we selected at random the answer to the i^{th} query *twice*, we got a dot both

[4]For ease of reference, we report Cauchy-Schwartz inequality below:

$$(\forall N \in \mathbb{N})(\forall a_1, \ldots, a_N \geq 0).\Big[\sum_i a_i^2 \geq 1/N \Big(\sum_i a_i \Big)^2 \Big]$$

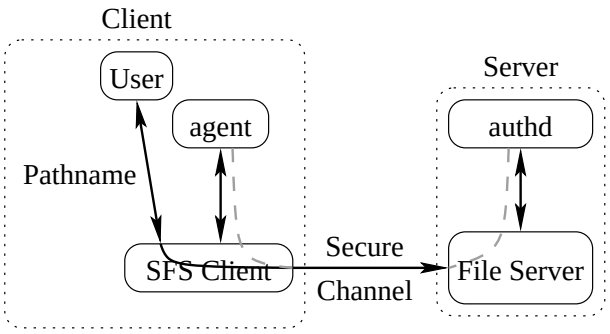

Figure 2.3: SFS user-authentication architecture.

times and the answers chosen—e_1 and e_2—were different. Indeed, since i and J are fixed, in this case A succeeds twice and uses the same important/critical queries. Since these two runs are now *independent*, the needed probability is simply computed as:

$$\delta_{i,J,R} = \varepsilon_{i,J,R}(\varepsilon_{i,J,R} - 2^{-\ell}) = \varepsilon_{i,J,R}^2 - \varepsilon_{i,J,R}2^{-\ell} \tag{2.3}$$

Finally, by conditioning $\delta_{i,J}$ on the value of R, using Equations 2.2, (2.3), and Cauchy-Schwartz again, we get:

$$\begin{aligned}\delta_{i,J} &= \frac{1}{|R|}\sum_R \delta_{i,J,R} = \frac{1}{|R|}\Big(\sum_R \varepsilon_{i,J,R}^2 - \sum_R \varepsilon_{i,J,R}2^{-\ell}\Big) \\ &= \Big(\frac{1}{|R|}\sum_R \varepsilon_{i,J,R}^2\Big) - \frac{\varepsilon_{i,J}}{2^\ell} \geq \Big(\frac{1}{|R|}\sum_R \varepsilon_{i,J,R}\Big)^2 - \frac{\varepsilon_{i,J}}{2^\ell} \\ &= \varepsilon_{i,J}^2 - \varepsilon_{i,J}2^{-\ell}\end{aligned}$$

Equation (2.1), and hence the overall bound, follows. □

2.4 Implementation

Figure 2.3 illustrates the major components of SFS involved in user authentication. The file system client and file server communicate over a TCP connection,

encrypting and MACing all traffic to obtain a secure channel. User authentication itself is actually performed by processes external to the file system. On the client, every user runs an *agent* program responsible for authenticating it to remote servers. On the server side, a program *authd* is responsible for validating authentication requests and translating them into credentials meaningful to the file server.

When a user accesses a file server for the first time, the file system client delays the access and asks the user's agent to authenticate her to the server. The agent then communicates with the server's authd to obtain appropriate privileges for the user. The agent and authd communicate through the file system's secure channel, but the file system views their messages as opaque byte arrays. Thus, new authentication protocols can be implemented without modifying the file system software.

Figure 2.4 shows the interface between the file system, agent, and authd. Every secure channel between a client and server is identified by a unique session ID, *SessID*. SessID, when hashed together with the server's name, public key, and certain other information, produces a value called *AuthID*. When the SFS client asks a user's agent to authenticate her to a server, it sends the agent the SessID of the session with that server, a sequence number, *Seq#*, identifying the authentication request within that session, and several other pieces of information including the name of the server. The agent computes AuthID and then communicates with the server's authd. If the authentication protocol succeeds, the authd informs the file server of the user's Seq#, AuthID, and credentials. The server then returns a short handle *Auth#* to the client, which the client subsequently uses to tag all file system requests on behalf of that user.[5]

[5]SFS could equally well have chosen to tag requests with Seq#, but Auth# is a shorter

AuthID = SHA-1(SessID, Server, . . .)

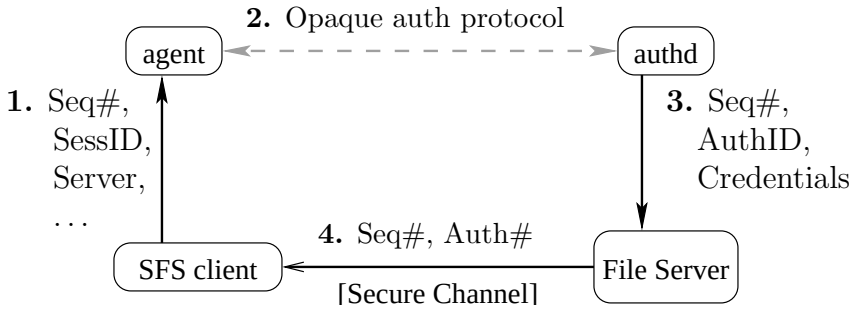

Figure 2.4: Messages exchanged during the user-authentication process. The authentication protocol between the agent and authd is opaque to the core file system software.

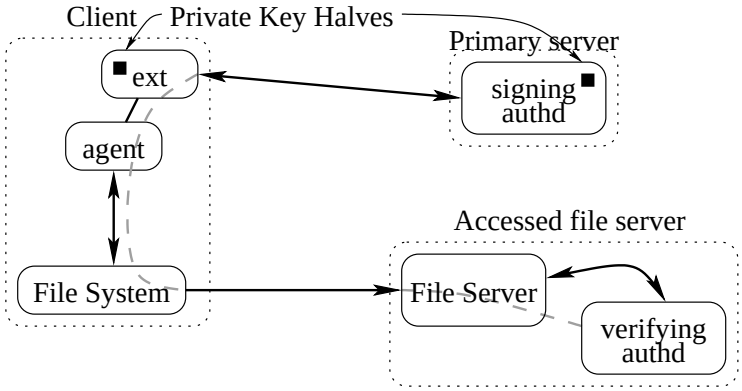

Figure 2.5: Implementation of proactive signatures in SFS

In the original SFS authentication system, authd keeps a mapping of users' public keys to credentials, while the agent keeps one or more private keys in memory. The authentication protocol consists of the user digitally signing {Seq#, AuthID}. The original protocol used Rabin-Williams [78] digital signatures.

In addition to validating file server users, SFS's authd plays a separate role as a repository of users' encrypted private keys. SFS users can store encrypted copies of their private keys with the authd of their "primary" SFS server. After logging into a client machine, users typically connect to their primary server's authd over the network, authenticate themselves through the SRP [79] secure password protocol, and then retrieve their encrypted private keys. (Users also end up securely downloading server public keys this way; *cf.* [68] for details.)

2.4.1 Implementing 2Schnorr in SFS

Integrating 2Schnorr in SFS was relatively straight-forward, as the original user authentication protocol already consisted of a simple digital signature on {Seq#, AuthID}.

On the server side, we made several modifications to authd. We extended it to support both Schnorr and Rabin public keys. We modified the server's encrypted private-key repository functionality, so that it now optionally holds both an encrypted half of a user's private key and an unencrypted one. We added an option to the RPC by which users update their login information so as to update the two key halves whenever users change their passwords. Finally, we added a SIGN RPC that implements the server side of the two-message

and therefore slightly more convenient value.

2Schnorr protocol.

In order to access the new SIGN RPC, a user must first authenticate himself to the server. The simplest way is through SRP. When a client downloads a user's encrypted private key half, it is permitted to keep the connection open to the server for issuing SIGN requests. The server is currently willing to endorse two types of message—login requests, and requests to change the user's public key on a particular server. Both types of messages include an AuthID, which authd computes and verifies. Computing AuthID involves hashing, among other things, the name and public key of the server being accessed and the type of service being requested (remote login, file server, *etc.*). Authd logs this information, leaving a complete audit trail in case an attacker steals a user's password.

On the client side, rather than hard-code 2Schnorr into the agent, we instead implemented an extension facility by which arbitrary external programs can plug into the agent and offer to attempt user-authentication. Figure 2.5 illustrates the complete system. Upon loading the 2Schnorr private key half, an external authentication process *ext* plugs into the agent, keeping open a connection to the user's primary authd, which we call the *signing authd*. When the user accesses a new file server, the agent queries the ext process, which executes 2Schnorr with the signing authd to produce an ordinary Schnorr signature. The *verifying authd* on the server that the user is accessing then verifies the Schnorr signature to authenticate the user.

Several other implementation details are worth mentioning. The new authd can actually store two private keys for a user. This is important so that a user who changes her public key can access both the old and new private keys for a time. On the client side, while ext is waiting for the server to endorse a

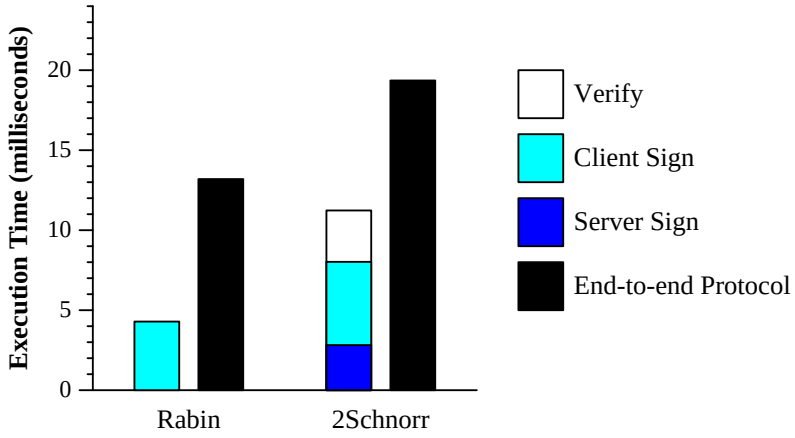

Figure 2.6: Benchmarks for signing and verifying in the Rabin and Schnorr signature schemes. End-to-end protocol shows user wait time for complete authentication.

signature, it precomputes $g^{k_c} \bmod p$ for the next signature, to reduce latency. Also, in order to compute the value $s_c = k_c + x_c e \bmod q$, the client actually computes $s_c = (k_c\,(e^{-1} \bmod q) + x_c)\,e \bmod q$ to thwart any timing attacks based on non-constant time of the modular reduction. s_s is computed similarly.

2.5 Performance

This section evaluates the performance of 2Schnorr and its impact on SFS. The two most important effects of 2Schnorr are on the responsiveness of the client and on CPU consumption on the server. In both cases, we compare the new 2Schnorr authentication protocol to the original SFS authentication protocol, which is based on an optimized, non-interactive Rabin signature scheme. We show that the new protocol has little impact on system responsiveness for clients with good network conditions. It is more expensive on the server-side in

comparison to Rabin, but still cheap in an absolute sense.

We measured the 2Schnorr and Rabin algorithms both in isolation and as part of a file system access that required user authentication. We used three separate machines in our experiments: a *verifying-authd* server, a *signing-authd* server, and a client. The *verifying-authd* machine served the file system we used in the file access benchmark, while the *signing-authd* performed the 2Schnorr server-side protocol (and hence was not used in the Rabin experiments). We used 1.75 GHz Athlon computers for both servers, and a 1.4 GHz Athlon for the client. All three machines had sufficient memory so that no paging activity was detected during any of the trials. The three machines were connected by switched 100 Mbit ethernet, with round trip latencies below 0.2 ms between every pair of machines. The *verifying-authd* was running FreeBSD 4.6.2, while the *signing-authd* and client were running OpenBSD 3.1. All machines used GMP version 3.1.1 for large integer arithmetic.

The experiments were conducted using Rabin keys with a 1024-bit modulus, and Schnorr keys with 1024-bit ps and 160-bit qs. There are no known efficient reductions from the discrete log problem to factorization or vice-versa. However, given today's fastest algorithms, taking discrete logs over the group in the Schnorr algorithm should be roughly comparable to factoring the Rabin modulus.

To measure system responsiveness, we timed a **cd** command on the client to a directory (on the *verifying-authd*) that triggered an authentication. Under normal circumstances, a user is authenticated to a remote SFS server as long as that server is mounted and the user does not add or remove keys from his agent. In our experimental setup, however, we reset the user's agent after every successful authentication. The results of this experiment are shown by the

black bars in Figure 2.6. Without network latency, the 2Schnorr protocol is 47% slower. However, in absolute terms, 2Schnorr is only 6 msec slower, which is a barely noticeable delay for the first access to a file system. In fact, when the file system client is not already connected to the server, there is additional time to connect and negotiate a session key, which further reduces the relative difference of Rabin and 2Schnorr. On the other hand, had there been greater latency between the client and *signing-authd*, the 2Schnorr authentication time would increase by the network round trip time.

Figure 2.6 also shows the CPU times required to compute and verify digital signatures. Note that verifying in Rabin is negligible, as no modular exponentiation is required. The *verifying-authd* can verify a Rabin signature in well under 0.1 msec. By contrast, Schnorr signature verification takes approximately 3 msec—a significant increase. For this reason, Schnorr might be a bad candidate for a *verifying-authd* server that supported huge numbers of users with high turnover. However, since every client connection also requires the server to engage in the key negotiation protocol, SFS servers cannot scale to 1,000s of new connections per second anyway.

If we compare the cost of signing, the sum of the two halves of the 2Schnorr signature protocol is 87% slower than Rabin. As 2Schnorr overlaps its calculation of $g^{k_c} \bmod p$ with network latency and server computation, the cost of this computation, about 1.7 msec, is not reflected in the CPU times shown in the graphs.

Finally, we should note that key generation with 2Schnorr is significantly slower. We generated keys on the *signing-authd* and found that Rabin keys can be generated in about 0.2 seconds, while 2Schnorr keysets require about 0.55 seconds. Because users will rarely need to regenerate keys, this slowdown is

acceptable.

Though 2Schnorr is clearly more expensive than the original Rabin-based user authentication scheme, the performance is still perfectly acceptable for a procedure that only needs to be invoked when a user first accesses a new file server. Moreover, we believe the performance impact is more than offset by the 2Schnorr's added security.

2.6 Summary

This chapter develops proactive, two-party signature schemes (P2SS) as an effective tool to address the challenges of user-authentication in settings with many administrative realms. We present a three-message protocol, 2Schnorr, which is provably secure in the random oracle model assuming only the difficulty of the computational discrete log problem. For systems with a constant bound on the number of concurrent signature requests, we also give a two-message version of 2Schnorr, which we prove secure using the stronger one-more-discrete-log assumption. We argue that similar techniques can be used for a P2SS version of the GQ signature scheme.

To demonstrate the utility of P2SS, we integrated 2Schnorr into SFS, a secure network file system. Using 2Schnorr, a user whose password is compromised can recover by simply changing his password on his primary server. This will immediately block attackers from accessing his accounts in all other administrative realms where he has registered the same public key. Moreover, the user can also obtain from his primary server a log of all servers accessed by the attacker—possibly including accounts the user has forgotten about. While 2Schnorr is slower than SFS's original Rabin signature algorithm, we show that

the performance impact is quite acceptable, particularly given the added security.

Chapter 3

E-Mail Authentication via Social Networks

A variety of peer-to-peer systems use social networks to establish trust between participants. Yet the sharing of social information introduces privacy concerns. This chapter describes new privacy-preserving cryptographic protocols that enable participants to verify social proximity while exposing minimal information about the parties' social contacts. Compared to previous results, our protocols are either significantly more efficient (orders of magnitude faster than the private-matching approached used in PM [50]) or achieve stronger security properties at similar cost.

3.1 Introduction

In peer-to-peer systems where resources are scarce or users are subject to abuse, participants can leverage social relationships to guide their interactions with other users. Further considering transitive trust relationships can extend a

user's vantage, while still incurring a low risk of coming across abusive users. In the email or instant messaging contexts, for example, social networks can facilitate cooperative spam blacklisting [64] or sender whitelisting [50].

A naïve approach to discover transitivity is for one party to send his list of friends to the other party, who computes the set intersection of their two input sets. Yet this simple form of information sharing introduces privacy concerns.

While the problem of privacy-preserving two-party computation has been widely studied in the cryptographic literature [80, 52], general-purpose cryptographic solutions are too computationally expensive for practical use. Furthermore, their privacy guarantees are often misaligned with applications' specific threat models (discussed in Section 3.3).

This chapter describes efficient cryptographic protocols with which parties can determine shared friends while exposing minimal information about their social contacts. Using Re: [50] as a motivating example—an email system that reliably accepts mail from senders based on proximity in a social network—we describe two alternative methods to verify social proximity. The first method, based only on cryptographic hash functions and symmetric encryption, meets all of Re:'s current privacy and security goals at a fraction of the cost of its current Private Matching [48] protocol. The second method, while of comparable cost, achieves stronger privacy guarantees (namely, *non-transferability*) through its novel use of cryptographic properties of bilinear groups [17].

Our contributions are twofold. First, we describe and define a security model for verifying social connectedness in a privacy-preserving fashion (Section 3.3). In fact, the mismatch between Re:'s goals and the privacy properties offered by Private Matching were a source of both computational inefficiency and privacy limitations. Second, we propose cryptographic protocols that protect such social

proximity queries, for both scenarios that require high efficiency (Section 3.4.1) and those that demand strong security properties (Section 3.4.2).

3.2 Motivating application: Re:

Reliable Email (Re:) [50] is an automated email acceptance system that whitelists email according to its sender. It seeks to undue the email unreliability introduced by content-based filters and other spam-fighting technologies which, while seeking to minimize the amount of spam that reaches a user's inbox, occasionally misclassify legitimate mail as spam.

The concept of sender-based whitelisting for email is hardly new. Yet, traditional whitelists suffer from two chief usability issues. First, a recipient's whitelist cannot accept mail from a sender previously unknown to the recipient. Second, populating whitelists requires manual effort distributed diffusely in time, as users acquire new contacts.

To overcome these limitations, Re: *automatically* broadens the set of senders whose mail is accepted by recipients' whitelists by explicitly examining the social network among email users. Specifically, Re: allows a user R to *attest* to another user S, which indicates that R is willing to have email from S directly forwarded to his mailbox. In other words, "User R trusts his *friend* S not to send him spam." Such an attestation is a digitally-signed statement of the form:[1]

$$\sigma_{R \to S} = \{H(R), H(S), start, end\}_{SK_R}$$

where H is a collision-resistant cryptographic hash function like SHA-256 operating on the users' email addresses, *start* and *end* define the attestation's

validity period, and SK_R denotes user R's signing key.

RE: leverages these attestations for accepting mail in cases where the sender S and recipient R are *not* already friends, but instead share a *bridging* friend T, resulting in a *friend-of-friend* (FoF) relationship between S and R.

By performing an FoF query, a recipient can determine which of his friends, if any, have attested to the sender. RE: achieves this while still providing the following privacy properties:

- The sender S does not learn anything about R's friends. Both learn an upper bound on the number of friends presented by the other, however.
- The recipient R learns only the intersection of the two sets of friends, *i.e.*, those T for whom R signed $\sigma_{R \rightarrow T}$ and from whom S received $\sigma_{T \rightarrow S}$.
- A third party observing all messages between S and R learns an upper bound on the size of each input, but nothing about their content nor the intersection size.
- Only R can execute the FoF query.

RE: provides the final property through its use of a one-time authorization token, while the first three properties are achieved through the use of a Private Matching (PM) protocol [48].

At a high level, PM is a two-party interactive protocol, where the input of each party is a set, and the output (learned only by one party) is the input sets' intersection. PM uses the homomorphic properties of certain public-key encryption schemes.

[1]The original notation used in RE: [50] for attestations had the form $R \rightarrow S$; we chose to adopt a subscripted notation to reserve "plain arrows" to denote social links (see Section 3.3).

In RE:'s case, R's inputs are the email addresses of those X such that $\sigma_{R \rightarrow X}$, while S's inputs are those Y such that $\sigma_{Y \rightarrow S}$, along with the $\sigma_{Y \rightarrow S}$ themselves as payloads for each input. After running PM, R learns the email addresses for the set of bridging friends $\mathcal{T}$ and the corresponding attestations $\{\sigma_{T \rightarrow S} : T \in \mathcal{T}\}$. R finally verifies the digital signatures on these attestations before whitelisting S's email.

RE:'s initial concern with sharing friendship lists ("address books") for whitelisting purposes was the potential for spammers to use such a mechanism to harvest valid email addresses. RE:'s use of the PM protocol certainly prevents such an attack. It does not, however, prevent parties from "lying" about their inputs,[2] *e.g.*, by including in their input sets email addresses of people for whom they do not have the appropriate attestations.

While in this context the sender S cannot benefit from lying—as R will check the recovered attestations' signatures, match them to the supplied email addresses, and verify that the proper attestation path exists—a deceitful recipient may lie to mount a targeted attack against those parties that consider S a friend, for example. Namely, to verify whether some party Z considers S a friend, R simply claims to consider Z a friend himself when performing an FoF query with S: if S has an attestation $\sigma_{Z \rightarrow S}$, R will receive such attestation as part of PM's output.

Within RE:'s ill-defined security model, it is not even clear if and how such behavior could be construed as a protocol abuse, as R can generate an attes-

[2]Indeed, private function evaluation protocols, of which PM is an instance, are proven secure in a model that only concerns itself with preventing information leakage from the function's execution; the model does not embed a notion of "proper" inputs, so one cannot directly reason about lying parties.

tation $\sigma_{R \to Z}$ at any time anyway (say, with a very short duration). In the next section, we propose a more formal privacy model for verifying proximity in social networks that directly addresses these shortcomings.

3.3 Model

A social network can be modeled as a directed graph $G = (V, E)$, whose vertices represent the users of the system and where the presence of an arc $(R, T) \in E$ (also denoted $R \to T$) indicates the existence of a social relationship between user R and user T. We will discuss the implications attached to such relationships shortly; for now, we will just take $R \to T$ to mean that "T is R's friend."

This graph is represented within the system in a distributed fashion: each participant has only a local view of the network, consisting of its incoming and outgoing arcs. Additionally, the system provides a *proximity check* mechanism by which a user S can help R determine whether he is "close enough" to her in the social network. In particular, R can find out all *bridging* friends X such that $R \to X$ and $X \to S$. Such mechanism is exposed to a higher-level application, in which users send *requests* to each other, and requests may be treated differently by the recipient according to the social proximity of the sender (*e.g.*, whitelisting FoF's in Re:).

Figure 3.1 illustrates this for a fragment of a social network, where R learns that there is exactly one bridging friend between him and S, namely T. Notice that both T and W are directly connected to S, but R should not learn about W since the arc $R \to W$ does not appear in the graph.

To properly address privacy concerns of this kind, we first elaborate on the

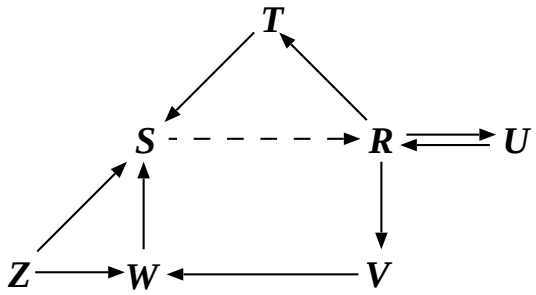

Figure 3.1: A fragment of a social network. Solid arrows represent trust relationships; the dotted arrow highlights a pair of users for which to verify social proximity.

nature of the relationships represented by the social network. In the context of Re:, such relationships were viewed as predominantly unidirectional: $R \to T$ roughly corresponds to the notion that "user R trusts T not to send him spam." Under this interpretation, whether the arc $R \to T$ appears in the social network or not is essentially up to R. As we alluded in Section 3.2, however, this approach is arguably too lax: Building on the example of Figure 3.1, an overly curious R could unilaterally augment the social network with arcs $R \to U$, $R \to W$, $R \to Z$. This would "entitle" R to learn about the social link $W \to S$ when receiving email from S, breaching the privacy of both W and S.

To this effect, we posit that the presence of social link $R \to T$ ought to express consent of *both* parties:

(Forward Trust) User R places some form of trust on user T that T can use to demonstrate to some U the presence of a chain $U \to R$, $R \to T$.

(Backward Authorization) User T authorizes user R to discover links of the form $T \to X$ when trying to establish the existence of a social chain such

as $R \to T$, $T \to S$.

Within a specific system, each such requirement would be associated with some concrete piece of data. For example, R's trust in T could be expressed via a digitally-signed attestation *à la* Re:, whereas backward authorization could be implemented as a shared secret key that T gives to R (as in Section 3.4).

Under such a setup, one can formalize a system's privacy properties by explicitly pointing out what information is exposed to the users, in terms of guarantees of the form: "During a proximity check with user S, user R learns at most $\mathcal{I}$." Following the approach of secure multi-party computation [80, 52], a statement of this sort is proved by showing that, given $\mathcal{I}$ and the knowledge held by R (which can be deduced from R's social relationships), it is possible to simulate (or "fake") the content of all messages seen by R during the proximity check. This implies that any other information exposed to R can be derived using only $\mathcal{I}$ and R's knowledge. Thus, $\mathcal{I}$ itself provides an upper bound on the extra knowledge that R gains. We apply this proof technique in Section 3.4 to assess the privacy of our constructions.

3.4 Constructions

3.4.1 An Efficient Hash-Based Construction

Our first construction assumes, as in Re:, a signing/verification key pair SK_R/VK_R for each user R. Additionally, a user R maintains a secret seed s_R for a cryptographic pseudo-random function $\mathcal{F}$ like HMAC-SHA-256.

Each arc in the social network is associated with a (pseudo-)random key, termed the arc's a-value. All a-values corresponding to arcs of the form $R \to X$

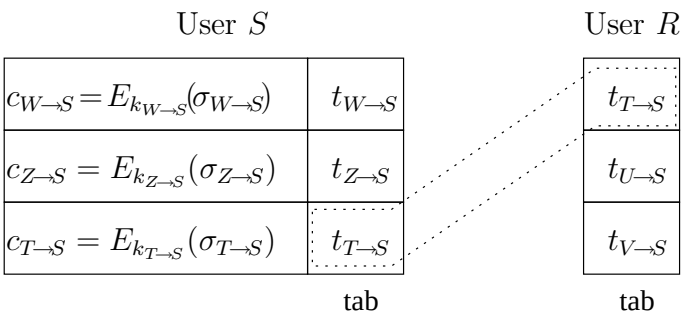

Figure 3.2: Data structures used for a hash-based proximity check between a sender S and recipient R.

are derived from R's secret seed s_R as: $a_{R\to X} = \mathcal{F}_{s_R}(\text{“arc”}, R, X)$.

For each social link of the form $R \to X$, user R creates an attestation $\sigma_{R\to X}$ for user X, and sends it to X along with $a_{R\to X}$ (*forward trust*). In return, R receives s_X from X (*backward authorization*).

This asymmetry in exchanging secrets stems from the way we implement proximity checks: Roughly speaking, for Y such that $Y \to S$, the sender S encrypts the attestation $\sigma_{Y\to S}$ under (a key derived from) $a_{Y\to S}$. In turn, for X such that $R \to X$, the receiver R tries to read these encrypted attestations using (a key derived from) the a-value $a_{X\to S}$ (corresponding to the possibly non-extant arc $X \to S$), which R can compute given s_X.

To help R in his decryption process (which, as described, requires a quadratic amount of symmetric-key operations), S includes a *tab* $t_{Y\to S}$ along with each encrypted attestation $c_{Y\to S}$ (*cf.* Figure 3.2). More in detail, for each arc $Y \to S$, S combines the attestation $\sigma_{Y\to S}$ and the a-value $a_{Y\to S}$ into a *tabbed encrypted attestation* $(c_{Y\to S}, t_{Y\to S})$ as follows. The *tab* $t_{Y\to S}$ is a pseudo-random hash computed under $\mathcal{F}$ keyed with $a_{Y\to S}$ *i.e.*, $t_{Y\to S} = \mathcal{F}_{a_{Y\to S}}(\text{“tab”}, \mathit{ReqID})$, where *ReqID* is a unique identifier supplied by the higher-level application.

The ciphertext $c_Y = \mathcal{C}_{k_{Y\to S}}(\sigma_{Y\to S})$ is computed under a secure symmetric cipher $\mathcal{C}$ (*e.g.*, AES-CBC), with a key $k_{Y\to S}$ also derived from $a_{Y\to S}$: $k_{Y\to S} = \mathcal{F}_{a_{Y\to S}}(\text{“key”}, \mathit{ReqID})$.

At this point, S creates a list of these tabbed encrypted attestations, one for each of her incoming social relationships, permutes this list in random order, and sends it to R along with her request.

User R processes such a list by first looking at the tab component of each entry. In particular, for each relationship of the form $R \to X$, R holds the seed s_X. So R can form the a-value $a_{X\to S} = \mathcal{F}_{s_X}(\text{“arc”}, X, S)$, and then the $\mathcal{F}$-hash of *ReqID* (which was included as part of S's request) under $a_{X\to S}$. In this way, R computes his own set of tabs, and compares them with those received from S (which can be done efficiently, *e.g.*, by first storing one set of tabs in a hash-table, and then trying to retrieve from it the tabs of the other set). Thanks to the cryptographic properties of $\mathcal{F}$, it is extremely unlikely that two such tabs will coincide, except when they are created from the same seed. In other words, a match between the tabs guarantees that the same seed was used by both R and S, which in turn reveals the bridging friend(s), say T. At this point, R can compute the proper key $k_{T\to S} = \mathcal{F}_{a_{T\to S}}(\text{“key”}, \mathit{ReqID})$ and decrypt the corresponding encrypted attestation, thus recovering $\sigma_{T\to S}$. Finally, R verifies T's signature on $\sigma_{T\to S}$ before concluding that $R \to T$ and $T \to S$.

Security proof. Clearly, malicious senders do not pose any privacy threat, because the protocol consists just of a single sender-receiver flow. As for a malicious receiver R, we now prove that he only learns how many friends have attested to S and those attestations for which the attester is a common friend *i.e.*, $\mathcal{I} = (|\mathcal{Y}|, \{\sigma_{X\to S} : X \in \mathcal{T}\})$, where $\mathcal{Y} = \{Y : Y \to S\}$ and $\mathcal{T} = \{X : R \to X, X \to S\} \subseteq \mathcal{Y}$. To this end, we need to show how to simulate the message

that R receives from S, given $|\mathcal{Y}|$, $\{\sigma_{X\to S} : X \in \mathcal{T}\}$ and the shared secrets known to R.

We start by observing that for any $W \in \mathcal{Y} \setminus \mathcal{T}$, W's random seed s_W is unknown to R, so that $a_{W\to S}$ is (pseudo-)random in R's view. Hence, by the properties of pseudo-random functions [51], it is infeasible to tell $k_{W\to S} = \mathcal{F}_{a_{W\to S}}(\text{“key”}, \mathit{ReqID})$ (resp. $t_{W\to S} = \mathcal{F}_{a_{W\to S}}(\text{“tab”}, \mathit{ReqID})$) apart from a random string $\tilde{k}_{W\to S}$ (resp. $\tilde{t}_{W\to S}$) of the same length. It follows that no efficient algorithm can distinguish $c_{W\to S} = \mathcal{C}_{k_{W\to S}}(\sigma_{W\to S})$ from $\mathcal{C}_{\tilde{k}_{W\to S}}(\sigma_{W\to S})$, which in turn, since $\mathcal{C}$ is a secure symmetric encryption scheme, cannot be distinguished from $\tilde{c}_{W\to S} = \mathcal{C}_{\tilde{k}_{W\to S}}(0^{|\sigma_{W\to S}|})$. Thus, we can replace $(c_{W\to S}, t_{W\to S})$ in S's message with a “randomized” pair $(\tilde{c}_{W\to S}, \tilde{t}_{W\to S})$, without R noticing the change.

Simulating the tabbed encrypted attestation $(c_{T\to S}, t_{T\to S})$ for $T \in \mathcal{T}$ is easier, since in this case we have $\sigma_{T\to S}$ (from $\mathcal{I}$) and $a_{T\to S}$ (as $R \to T$, and so, by backward authorization, R knows s_T, from which $a_{T\to S}$ is derived). Thus, we can directly compute $k_{T\to S} = \mathcal{F}_{a_{T\to S}}(\text{“key”}, \mathit{ReqID})$, $c_{T\to S} = \mathcal{C}_{k_{T\to S}}(\sigma_{T\to S})$ and $t_{T\to S} = \mathcal{F}_{a_{T\to S}}(\text{“tab”}, \mathit{ReqID})$.

3.4.2 Privacy in the Face of Collusions

Compared to the privacy properties of the PM-based protocol of Re:, our hash-based construction additionally guarantees that receivers cannot learn about attestations created by a user T without T's permission. This is in keeping with the notion of backward authorization, an aspect of our modeling missing from Re:'s original framework.

However, the kind of backward authorization implemented by the hash-based scheme is *transferable*: if user T authorizes user R to learn about attestations

of the form $\sigma_{T \rightarrow X}$, R can further transfer such authorization to another user U. Then, during a proximity check with S, U would be able to discover the attestation $\sigma_{T \rightarrow S}$, even though the social link $U \rightarrow T$ is absent and so U was never back-authorized by T.

Notice that this scenario does not contradict the privacy guarantees proved in Section 3.4.1; rather, it points out the privacy implications that collusions of two or more users can have. In fact, it is unclear whether this ought to be considered a privacy problem: After all, if R and U pool their resources together, then they appear as "one and the same" to the rest of the system. Since a proximity check between S and R would have disclosed $\sigma_{T \rightarrow S}$ anyway, we may deem that U learning $\sigma_{T \rightarrow S}$ is a reasonable outcome.

In settings where user collusions are of concern, however, we may want to attain *non-transferability*: namely, only enable those users that T has *individually* authorized to actually learn about his attestations. We now describe a construction that leverages the cryptographic properties of bilinear groups to satisfy this stronger requirement.

Bilinear groups are pairs of cryptographic groups $\mathcal{G}_1$ and $\mathcal{G}_2$ of the same order q (for some large prime q), equipped with an efficiently computable map $e : \mathcal{G}_1 \times \mathcal{G}_1 \rightarrow \mathcal{G}_2$ such that $e(g^a, h^b) = e(g, h)^{ab}$ for all $g, h \in \mathcal{G}_1$ and all $a, b \in \mathbb{Z}_q$ (*bilinearity*).[3] Typical examples of bilinear groups are based on elliptic and hyperelliptic curves (*e.g.*, [61, 17]).

Our bilinear construction exploits the bilinearity of the e map to enable users to "personalize" the secret values that they give out for backward authorization when establishing a social link (whereas in the hash-based scheme of

[3]Technically, the map should also be *non-degenerate*: not all pairs in $\mathcal{G}_1 \times \mathcal{G}_1$ should map to the unit in $\mathcal{G}_2$.

Section 3.4.1, user R gives out the same secret s_R to all X such that $X \to R$). In particular, each user R maintains a secret exponent $s_R \in \mathbb{Z}_q$ and a public value $y_R^{\text{own}} = H_1(\text{“own”}, R)^{s_R} \in \mathcal{G}_1$, where $H_1 : \{0,1\}^* \to \mathcal{G}_1$ is a random oracle [9] with range in $\mathcal{G}_1$.[4] Then, R hands out $y_{R \to X}^{\text{fwd}} = H_1(\text{“fwd”}, R, X)^{s_R}$ to each X for which $R \to X$, and $y_{X \to R}^{\text{bwd}} = H_1(\text{“bwd”}, X, R)^{s_R}$ to those X for which $X \to R$. Notice that the bilinear property enables X to verify the correctness of the value received from R, since for properly computed $y_{R \to X}^{\text{fwd}}$, it must hold that $e(H_1(\text{“own”}, R), y_{R \to X}^{\text{fwd}}) \stackrel{?}{=} e(y_R^{\text{own}}, H_1(\text{“fwd”}, R, X))$, and a similar check can be performed to test the correctness of $y_{X \to R}^{\text{bwd}}$.

The proximity check protocol between S and R uses the same overall structure as that of the hash-based scheme, except that the a_* seeds for the pseudo-random function $\mathcal{F}$ are now computed as follows: For each $Y \to S$, S sets $a_{R,Y,S} = e(y_{Y \to S}^{\text{fwd}}, H_1(\text{“bwd”}, R, Y))$. Then, S can compute a tabbed encrypted attestation as before, using $a_{R,Y,S}$ in place of $a_{Y \to S}$. R computes his tabs in a similar fashion for each $R \to X$, by setting $a_{R,X,S} = e(H_1(\text{“fwd”}, X, S), y_{R \to X}^{\text{bwd}})$. The only detail to check is that, for those T such that $R \to T$ and $T \to S$, both S and R obtain the same value $a_{R,T,S}$, which readily follows by bilinearity.

Security proof. One can show that this bilinear scheme preserves the privacy of S's email contacts even in the face of collusions. The proof follows the same approach as the one used for the hash-based scheme of Section 3.4.1; we omit the details, and only point out that the hardness assumption needed for the bilinear groups is the standard *Decisional Bilinear Diffie-Hellman Assumption* [17]: Given (g, g^a, g^b, g^c) for random $g \in \mathcal{G}_1$, $a, b, c \in \mathbb{Z}_q$, it is infeasible to distinguish $e(g,g)^{abc}$ from a random value in $\mathcal{G}_2$.

[4] Reliance on the random oracle model is not necessary, but we decided not to pursue alternative approaches for simplicity.

3.5 Discussion

Multi-Hop Proximity via Memoization. Although this paper has focused on friend-of-friend relationships, our hash-based protocol also supports a weak form of detection for longer social paths. Namely, we can build a multi-hop path $R \rightsquigarrow T$ and $T \rightsquigarrow S$, whereby $Y \rightsquigarrow X$ corresponds to a path of length $\ell \geq 1$ in which Y and X have directly authorized each other (*i.e.*, X knows s_Y and Y knows $a_{Y \to X}$), yet signed attestations only exists for pairs of adjacent users on the path $Y \rightsquigarrow X$, *i.e.*, $\sigma_{Y \to I_1}, \ldots, \sigma_{I_{\ell-1} \to X}$.

To use a social path from T of length $\ell > 1$, S encrypts the entire multi-hop attestation chain within the ciphertext $c_{T \to S}$ associated to $t_{T \to S}$. Note that this protocol does not prevent observers from learning an upper bound on the length of each encrypted chain.

Privacy *vs.* Auditability. In modeling the desired privacy guarantees, one could consider a more privacy-preserving definition: users only find out whether bridging friend(s) exist, not their actual identity. However, we argue that this stronger guarantee would limit the confidence that an application can place on social proximity: although social trust is transitive to an extent, it seems imprudent to assert that transitivity will always correctly predict trust relationships between bridged parties.

In Re:'s case, for example, a user might incorrectly attest to a spammer, or he might get compromised and begin acting as a spammer. By uncovering the identity of the linking friend, our protocols provide *auditability*, which helps coping with these scenario by enabling the decision-maker to review and correct the elements that led to the wrong decision.

Non-Interactive Implementation. Unlike the PM-based approach, both

methods described in Section 3.4 are non-interactive, requiring just a single message from S to R. This can significantly reduce system complexity (especially with respect to handling failures), and (for the specific case of Re:) facilitate integration with the existing e-mail infrastructure.

Symmetric Trust and Forward Security. For social networks with symmetric trust relationships (*i.e.*, where the links $Y \to X$ and $X \to Y$ are either both present in the social network, or both absent from the social network), our hash-based construction from Section 3.4.1 can be simplified by suppressing the a-values, and having the seed s_Y playing the role of $a_{Y \to X}$, for all of Y's social contacts X. Besides being conceptually simpler and computationally more efficient, this variant lends itself easily to extensions providing additional security properties, such as *forward security*, which we discuss next.

If a user S sends a request to U and no friend bridges U to S in the social network, all our constructions guarantee that U will not learn the identity of any of S's friends. Yet, as time passes and the social network evolves, a new social link may be established between U and one of S's friend (say, T). Now knowing s_T, if U has recorded S's request, he can recover T's earlier attestation to S.

Temporal correlations of this kind can be prevented in symmetric social networks by introducing time intervals in the model, and letting the s_* values evolve over time using hash chains. Namely, if the social link $T \to S$ is set up at time j_0, S gets from T the secret seed $s_T^{(j_0)}$. Then, at time j_1, S computes the tabbed encrypted attestation using the seed $s_T^{(j_1)}$ defined by the recurrence $s_T^{(i+1)} = H_s(s_T^{(i)})$, where H_s is a one-way permutation over the appropriate domain. Now, if U obtains $s_T^{(j_2)}$ from T at a later time j_2, he will not be able to use it to match the tabbed encrypted attestation that S included in her old

Party	Algorithm	Input sizes (number of friends) 10	100	1000
S	PM	589.7	27867.4	2490831.4
R	PM	14.7	110.9	1457.8
S	Hash	0.15	1.53	15.39
R	Hash	0.08	0.52	5.01

Table 3.1: Time (milliseconds) to perform privacy-preserving computations (with sender and recipient having inputs of the same sizes) for PM and hash-based protocols.

message, because doing so would require inverting H_s.

Performance Comparison. We now compare the performance of our hash-based construction (from Section 3.4.1) to the PM protocol used in Re:[50].

We instantiated the PM protocol using its faster ElGamal variant with 1024-bit keys. The hash-based construction uses HMAC-SHA-1 and AES-CBC with 128-bit keys. In both the PM and the hash-based schemes, attestations use 1024-bit Rabin signatures. Both microbenchmarks were performed on a 2.4-GHz AMD Athlon processor (in 32-bit mode) and do not include network overhead (which are nearly identical for both). Recipients in both protocols stopped analyzing results once three bridging friends were uncovered (a configurable parameter).

Table 3.1 reports the performance of the two protocols. As we see, the hash-based construction is orders of magnitude faster than the public-key-based PM protocol. This table does not include the time to verify any uncovered attestations, as it is identical in both (*e.g.*, $15\mu s$ per 1024-bit Rabin signature).

3.6 Summary

Peer-to-peer systems may use social networks in order to establish trust between participants, yet they introduce privacy concerns when sharing such information. In this paper, we define a privacy model for verifying social proximity. We use insights from this model to propose two cryptographic protocols that protect social proximity queries: a hash-based protocol that provides similar privacy to RE:'s proposed use of PM, yet is orders of magnitude faster; and a bilinear-groups-based protocol that introduces protection against collusion. We also described how to integrate our protocols with the Reliable Email whitelisting system. Our techniques are likely also applicable to other applications that leverage social networks.

Chapter 4

Authentication and Privacy for Group Access Control

This chapter introduces *Ad Hoc* Anonymous Identification schemes, a new multi-user cryptographic primitive that allows participants from a user population to form *ad hoc* groups, and then prove membership anonymously in such groups. Our schemes are based on the notion of *accumulator with one-way domain*, a natural extension of cryptographic accumulators we introduce in this work. We provide a formal model for *Ad Hoc* Anonymous Identification schemes and design secure such schemes both generically (based on any accumulator with one-way domain) and for a specific efficient implementation of such an accumulator based on the Strong RSA Assumption. A salient feature of our approach is that identification protocols take time independent of the size of the *ad hoc* group. All our schemes and notions can be generally and efficiently amended so that they allow the recovery of the signer's identity by an authority, if the latter is desired.

Via the Fiat-Shamir transform, we obtain *constant-size*, signer-ambiguous

group and ring signatures (provably secure in the Random Oracle Model). For ring signatures, this is the first such constant-size scheme, as all the previous proposals had signature size proportional to the size of the ring. For group signatures, we obtain schemes comparable in performance with state-of-the-art schemes, with the additional feature that the role of the group manager during key registration is extremely simple and essentially passive: all it does is accept the public key of the new member (and update the constant-size public key of the group).

4.1 Introduction

Anonymous identification is an oxymoron with many useful applications. Consider the setting, for a known user population and a known set of resources, where a user wants to gain access to a certain resource. In many cases, accessing the resource is an action that *does not* mandate positive identification of the user. Instead, it would be sufficient for the user to prove that he belongs to the subset of the population that is supposed to have access to the resource. This would allow the user to lawfully access the resource while protect his real identity and thus "anonymously identify" himself.

Given the close relationships between identification schemes and digital signatures, one can easily extend the above reasoning to settings where a user produces a signature that is "signer-ambiguous" *i.e.*, such that the verifier is not capable of distinguishing the actual signer among a subgroup of potential signers. In fact, it was in the digital signature setting that such an anonymous scheme was presented for the first time, with the introduction of the group signature model [29], which additionally mandates the presence of a designated

party able to reveal the identity of the signer, were the need to arise.

Subsequent work on group signatures and on anonymous identification in general [30, 36, 23, 28, 26, 35, 3, 1, 21, 24, 6, 2] allowed for more efficient designs and formal modelling of the primitive, with the current state of the art being the scheme by Ateniese et al.[1]. In general, existing group signature schemes are derived from their interactive counterpart (*ID Escrow* schemes [63]) via the Fiat-Shamir transform [42].

A related notion, but of slightly different nature, is that of ring signatures, introduced by Rivest, Shamir and Tauman in [73] and further studied in [22, 69]. Ring signatures differ from group signatures in that they allow group formation to happen in an *ad hoc* fashion: group must be formed without the help of a group manager; in fact, a user might not even know that he has been included in a certain group. This is in sharp contrast to the group signature setting where the user must execute a Join protocol with the group manager and obtain a group-membership certificate that cannot be constructed without the help of the group manager. Note that *ad hoc* group formation in the context of ring signatures is always understood within the context of a user population and an associated PKI. Based on the PKI, *ad hoc* subsets of the user population can be formed without the help of a "subset manager"—but it is assumed that every user has a registered public key.

While ring signatures are attractive because they have simple group formation procedures that can be executed by any user individually, they have the shortcoming that the length of the signature is proportional to the group size. For large groups, the length of a ring signature (growing linearly with the group size) will become impractical. To the contrary, schemes with *constant-size* signatures have been successfully designed in the group signature setting [1].

We remark that in the setting of anonymous identification, the counterpart of "signature size" is the bandwidth consumed by the protocol, which is thus an important complexity measure to minimize.

Based on the above discussion, an important open question in the context of anonymous identification and signature schemes, recently posed by Naor in [69], is the following:

> Is it possible to design secure anonymous identification schemes that enable *ad hoc* group formation in the sense of ring signatures and at the same time possess constant-size signature (or proof) length?

This chapter answers the above question in the affirmative. Specifically, we introduce a new primitive called *Ad Hoc* Anonymous Identification schemes; this is a family of schemes where participants from a user population can form groups in *ad hoc* fashion (without the help of a group manager) and then get anonymously identified as members of such groups.

Our main tool in the construction of *Ad Hoc* Anonymous Identification schemes is a new cryptographic primitive, *accumulator with one-way domain*, which extends the notion of a collision-resistant accumulator [12, 4, 25]. In simple terms, in an accumulator with one-way domain, the set of values that can be accumulated are associated with a "witness space" such that it is computationally intractable to find witnesses for random values in the accumulator's domain.

First, we demonstrate the relationship between such accumulators and *Ad Hoc* Anonymous Identification schemes by presenting a generic construction based on *any* accumulator with one-way domain. Second, we design an efficient implementation of accumulator with a one-way domain based on the Strong

RSA Assumption, from which we obtain a more efficient construction of *Ad Hoc* Anonymous Identification scheme whose security rests upon the Strong RSA Assumption.

We remark that previous work on anonymous identification that allowed subset queries was done by Boneh and Franklin [16]. They define a more limited security model, and show a protocol which imposes on both parties a computational load proportional to the subset size at each run. Moreover, their scheme is susceptible to collusion attacks (both against the soundness and against the anonymity of the scheme) that do not apply to our setting.

In our Strong-RSA-based *Ad Hoc* Anonymous Identification scheme, the computational and communication complexity on both ends is constant, regardless of the size of the group. Thus, the signature version of our *ad hoc* anonymous identification scheme yields a ring signature with constant size signatures (over a dedicated PKI). Other applications of our scheme include "*ad hoc* group signatures" (group signature schemes where the group manager can be offline during the group formation) and identity escrow over *ad hoc* groups.

Building on work by Camenisch and Lysyanskaya [25], Tsudik and Xu [77] investigated techniques to obtain more flexible dynamic accumulators, on which to base group signature schemes (which is one of our applications). The specific method used by [77] bears many similarities with our Strong-RSA-based instantiation, with some important differences. Namely, in their solution anonymity revocation takes time proportional to the user population, due to subtle problems concerning the accumulation of composite values inside the accumulator. Our work resolves this technical problem. Moreover, we present a new notion of *Ad Hoc* Anonymous Identification scheme, which has more applications than those specific to group signature schemes: for example, they allow us to build

the first constant-size ring signature schemes. We present a general construction for our primitives from any accumulator and not just the one of [25]. Our formal definitional framework is of independent interest.

4.2 Preliminaries

4.2.1 Notation

Throughout the chapter, we assume familiarity with the GMR notation [54], briefly summarized below.

A *negligible* function, denoted by $\mathsf{negl}(\lambda)$, is a function $f(\lambda)$ such that for all polynomials $p(\lambda)$, $f(\lambda) < 1/p(\lambda)$ holds for all sufficiently large λ.

An *efficient algorithm* $A(\cdot)$ is a probabilistic Turing machine running in expected polynomial time. An *adversary* $\mathcal{A}$ is a probabilistic, polynomial-time interactive Turing machine. If $A(\cdot)$ is an efficient algorithm and x is an input for A, then "$A(x)$" denotes the probability space that assigns to a string σ the probability that A, on input x, outputs σ. An efficient algorithm is *deterministic* if for every input x, the probability mass of $A(x)$ is concentrated on a single output string σ.

For a probability space P, "$x \stackrel{\mathrm{R}}{\leftarrow} P$" denotes the algorithm that samples a random element according to P. For a finite set X, "$x \stackrel{\mathrm{R}}{\leftarrow} X$" denotes the algorithm that samples an element uniformly at random from X. If $p(\cdot, \cdot, \ldots)$ is a boolean function, then "$Pr[x_1 \stackrel{\mathrm{R}}{\leftarrow} P_1, x_2 \stackrel{\mathrm{R}}{\leftarrow} P_2, \ldots \mid p(x_1, x_2, \ldots)]$" denotes the probability that $p(x_1, x_2, \ldots)$ is true after executing the algorithms $x_1 \stackrel{\mathrm{R}}{\leftarrow} P_1, x_2 \stackrel{\mathrm{R}}{\leftarrow} P_2, \ldots$.

A *two-party protocol* is a pair of interactive probabilistic Turing machines

(P, V). An execution (or run) of the protocol (P, V) on input x (for P) and y (for V) is an alternating sequence of P-rounds and V-rounds, each producing a message to be delivered to the other party (except for the last V-round). The sequence of such message is called the *transcript* of this run of the protocol. If, for all x and y, the length of such sequence, as well as the expected running time of P and V, are polynomial in the length of x and y, then (P, V) is an *efficient* two-party protocol. By "$(P(x) \leftrightarrow V(y))$", we denote the probability space that assigns to a sequence of strings π the probability that a run of the (P, V) protocol, on input x and y, will produce π as transcript.

4.2.2 NP-Relations and Σ-Protocols

An **NP**-relation R is a relation over bitstrings for which there is an efficient algorithm to decide whether $(x, z) \in R$ in time polynomial in the length of x. The **NP**-language L_R associated to R is defined as $L_R \doteq \{x \mid (\exists z)[(x, z) \in R]\}$

A Σ-protocol [33, 32] for an **NP**-relation R is an efficient 3-round two-party protocol, such that for every input (x, z) to P and x to V, the first P-round yields a *commitment* message COM, the subsequent V-round replies with a random *challenge* message CH, and the last P-round concludes by sending a *response* message RES. At the end of a run, V outputs a 0/1 value, functionally dependent on x and the transcript $\pi \doteq (\mathsf{COM}, \mathsf{CH}, \mathsf{RES})$ only; a transcript is *valid* if the output of the honest verifier is 1. Additionally, a Σ-protocol satisfies (1) *Special Soundness*, meaning that there is an efficient algorithm (called a *Knowledge Extractor*) that on input any $x \in L_R$ and any pair of valid transcripts with the same commitment message, $(\mathsf{COM}, \mathsf{CH}_1, \mathsf{RES}_1)$ and $(\mathsf{COM}, \mathsf{CH}_2, \mathsf{RES}_2)$ outputs z such that $(x, z) \in R$; and (2) *Special Honest-Verifier Zero-Knowledge*,

meaning that there is an efficient algorithm (called a *Simulator*) that on input $x \in L_R$ and any challenge message CH, outputs a pair of commitment/response messages COM, RES, such that the transcript $\pi \doteq (\mathsf{COM}, \mathsf{CH}, \mathsf{RES})$ is valid, and it is distributed according to the probability distribution $(P(x,z) \leftrightarrow V(x))$, for any z such that $(x,z) \in R$.[1]

The main result we will need about Σ-protocols is the following:

Theorem 6 ([53, 41])**.** *A Σ-protocol for any* **NP***-relation can be constructed if one-way functions exist.*

4.2.3 Accumulators

An *accumulator family* is a pair $(\{F_\lambda\}_{\lambda\in\mathbb{N}}, \{X_\lambda\}_{\lambda\in\mathbb{N}})$, where $\{F_\lambda\}_{\lambda\in\mathbb{N}}$ is a sequence of families of functions such that each $f \in F_\lambda$ is defined as $f : U_f \times X_f^{\mathsf{ext}} \to U_f$ for some $X_f^{\mathsf{ext}} \supseteq X_\lambda$ and additionally the following properties are satisfied:

- (efficient generation) There exists an efficient algorithm G that on input a security parameter 1^λ outputs a random element f of F_λ, possibly together with some auxiliary information a_f.
- (efficient evaluation) Any $f \in F_\lambda$ is computable in time polynomial in λ.
- (quasi-commutativity) For all $\lambda \in \mathbb{N}$, $f \in F_\lambda$, $u \in U_f$, $x_1, x_2 \in X_\lambda$,

$$f(f(u,x_1),x_2) = f(f(u,x_2),x_1)$$

[1] In particular, this implies that, for any $x \in L_R$ and any two z_1, z_2 such that $(x,z_1),(x,z_2) \in R$, the probability distributions induced by honest conversations between (i) a prover holding (x,z_1) and a verifier holding x; or between (ii) a prover holding (x,z_2) and a verifier holding x, are the same.

We will refer to $\{X_\lambda\}_{\lambda\in\mathbb{N}}$ as the *value domain* of the accumulator. For any $\lambda \in \mathbb{N}, f \in F_\lambda$ and $X = \{x_1, \ldots, x_s\} \subset X_\lambda$, we will refer to $f(\ldots f(u, x_1) \ldots, x_s)$ as the *accumulated value* of the set X over u: due to quasi-commutativity, such value is independent of the order of the x_i's and will be denoted by $f(u, X)$.

Definition 7. *An accumulator is said to be* collision resistant *if for any* $\lambda \in \mathbb{N}$ *and any adversary* $\mathcal{A}$*:*

$$Pr[f \xleftarrow{R} F_\lambda; u \xleftarrow{R} U_f; (x, w, X) \xleftarrow{R} \mathcal{A}(f, U_f, u) \quad | \\ (X \subseteq X_\lambda) \wedge (w \in U_f) \wedge (x \in X_f^{\mathsf{ext}} \setminus X) \wedge (f(w, x) = f(u, X))] = \mathsf{negl}(\lambda)$$

For $\lambda \in \mathbb{N}$ and $f \in F_\lambda$, we say that $w \in U_f$ is a *witness* for the fact that $x \in X_\lambda$ has been accumulated within $v \in U_f$ (or simply that w is a witness for x in v) whenever $f(w, x) = v$. We extend the notion of witness for a set of values $X = \{x_1, \ldots, x_s\}$ in a straightforward manner.

Accumulators with One-Way Domain. An *accumulator with one-way domain* is a quadruple $(\{F_\lambda\}_{\lambda\in\mathbb{N}}, \{X_\lambda\}_{\lambda\in\mathbb{N}}, \{Z_\lambda\}_{\lambda\in\mathbb{N}}, \{R_\lambda\}_{\lambda\in\mathbb{N}})$, such that the pair $(\{F_\lambda\}_{\lambda\in\mathbb{N}}, \{X_\lambda\}_{\lambda\in\mathbb{N}})$ is a collision-resistant accumulator, and each R_λ is a relation over $X_\lambda \times Z_\lambda$ with the following properties:

- (efficient verification) There exists an efficient algorithm D that on input $(x, z) \in X_\lambda \times Z_\lambda$, returns 1 if and only if $(x, z) \in R_\lambda$.
- (efficient sampling) There exists a probabilistic algorithm W that on input 1^λ returns a pair $(x, z) \in X_\lambda \times Z_\lambda$ such that $(x, z) \in R_\lambda$. We refer to z as a *pre-image* of x.
- (one-wayness) It is computationally hard to compute any pre-image z' of an x that was sampled with W. Formally, for any adversary $\mathcal{A}$:

$$Pr[(x, z) \xleftarrow{\mathrm{R}} W(1^\lambda); z' \xleftarrow{\mathrm{R}} \mathcal{A}(1^\lambda, x) \mid (x, z') \in R_\lambda] = \mathsf{negl}(\lambda)$$

4.2.4 The Strong RSA Assumption

We briefly review some definitions [12, 4] regarding the computational assumption underlying our efficient construction in Section 4.5.

A number n is an *RSA integer* if $n = pq$ for distinct primes p and q such that $|p| = |q|$. For $\lambda \in \mathbb{N}$, let RSA_λ be the set of RSA integers of size λ. A number p is a *safe prime* if $p = 2p' + 1$ and both p and p' are odd primes. A number n is a *rigid integer* if $n = pq$ for distinct safe primes p and q such that $|p| = |q|$. For $\lambda \in \mathbb{N}$, let Rig_λ be the set of λ-bit rigid integers.

Definition 8 (Strong RSA Assumption, [4])**.**
For any integer λ and for any adversary $\mathcal{A}$:

$$Pr[n \xleftarrow{R} \mathsf{Rig}_\lambda; z \xleftarrow{R} \mathbb{Z}_n^*; (x', y') \xleftarrow{R} \mathcal{A}(1^\lambda, n, z) \mid \big(y' > 1\big) \wedge \big((x')^{y'} \equiv z(n)\big)] < \mathsf{negl}(\lambda)$$

the probability being over the random choice of n and z, and $\mathcal{A}$'s random coins.

4.3 *Ad Hoc* Anonymous Identification Schemes

4.3.1 Syntax

An *Ad Hoc* Anonymous Identification scheme is a six-tuple of efficient algorithms (Setup, Register, Make-GPK, Make-GSK, Anon-ID$^{\mathsf{P}}$, Anon-ID$^{\mathsf{V}}$), where:

- Setup initializes the state of the system: on input a security parameter 1^λ, Setup creates a public database DB (that will be used to store information about the users' public keys), and then generates the system's parameters param; its output implicitly defines a domain of possible global parameters Par.

- Register, the *registration algorithm*, allows users to initially register with the system. On input the system's parameters param and the identity of the new user u (from a suitable universe of users' identity $\mathcal{U}$), Register returns a secret key/public key pair (sk, pk). To complete the subscription process, the user then sends his public key to a bulletin board for inclusion in a public database DB.

  The Register algorithm implicitly defines a domain $\mathcal{SK}$ of possible user secret keys and a domain $\mathcal{PK}$ of possible user public keys; its output induces a relation over user secret key/public key pairs, that we will denote by $\rightleftharpoons$. We also require a superset $\mathcal{PK}' \supseteq \mathcal{PK}$ to be specified, such that membership to $\mathcal{PK}'$ can be tested in polynomial time.

- Make-GPK, the *group public key construction algorithm*, is a deterministic algorithm used to combine a set of user public keys S into a single group public key gpk_S, suitable for use in the Anon-ID protocol described below. Syntactically, Make-GPK takes as input param and a set $S \subseteq \mathcal{PK}'$; its output implicitly defines a domain $\mathcal{GPK}$ of possible group public keys. We also require a superset $\mathcal{GPK}' \supseteq \mathcal{GPK}$ to be specified, such that membership to $\mathcal{GPK}'$ can be tested in polynomial time.

  The Make-GPK algorithm shall run in time linear in the number of public keys being aggregated; we also remark here that our definition forces Make-GPK to be *order-independent i.e.*, the order in which the public keys to be aggregated are provided shall not matter.

- Make-GSK, the *group secret key construction algorithm*, is a deterministic algorithm used to combine a set of user public keys S', along with a secret

key/public key pair (sk_u, pk_u), into a single group secret key gsk_u, suitable for use in the Anon-ID protocol described below.

Make-GSK takes as input param, a set $S' \subseteq \mathcal{PK}'$ and a key pair (sk_u, pk_u) satisfying $sk_u \rightleftharpoons pk_u$, and it shall run in time proportional to the size of S'. Its output implicitly defines a domain $\mathcal{GSK}$ of possible group secret keys.

The Make-GPK and Make-GSK algorithms can be used to extend the $\rightleftharpoons$-relation to $\mathcal{GSK} \times \mathcal{GPK}$, as follows: A group secret key $gsk \doteq \mathsf{Make\text{-}GSK}(\mathsf{param}, S', (sk, pk))$ is in $\rightleftharpoons$-relation with a group public key $gpk \doteq \mathsf{Make\text{-}GPK}(\mathsf{param}, S)$ if and only if $S = S' \cup \{pk\}$. Observe that even in the case that the $\rightleftharpoons$-relation is one-to-one over $\mathcal{SK} \times \mathcal{PK}$, it is usually many-to-one over $\mathcal{GSK} \times \mathcal{GPK}$, as more than one group secret key correspond to the same group public key.

- $\mathsf{Anon\text{-}ID} \doteq (\mathsf{Anon\text{-}ID^P}, \mathsf{Anon\text{-}ID^V})$, the *Anonymous Identification Protocol*, is an efficient two-party protocol, in which both $\mathsf{Anon\text{-}ID^P}$ (the *prover*) and $\mathsf{Anon\text{-}ID^V}$ (the *verifier*) get in input the system's parameters param and a group public key gpk (corresponding to some set S of user public keys *i.e.*, $gpk \doteq \mathsf{Make\text{-}GPK}(\mathsf{param}, S)$); $\mathsf{Anon\text{-}ID^P}$ is also given a group secret key gsk as an additional input.

  Any execution of the Anon-ID protocol shall complete in time independent from the number of public keys that were aggregated when constructing gpk and/or gsk; at the end of each protocol run, $\mathsf{Anon\text{-}ID^V}$ outputs a 0/1-valued answer.

Correctness. For correctness, we require that any execution of the Anon-ID protocol in which the additional input to $\mathsf{Anon\text{-}ID^P}$ is a group secret key gsk

$\rightleftharpoons$-related to the common input gpk, shall terminate with $\mathsf{Anon\text{-}ID}^{\mathsf{V}}$ outputting a 1 answer, with overwhelming probability. Formally:

$$
\begin{aligned}
&(\forall \lambda \in \mathbb{N})(\forall (u_1, \ldots, u_t) \in \mathcal{U}^*)(\forall m \in \mathcal{M}) \\
&\quad Pr[\mathsf{param} \xleftarrow{\mathrm{R}} \mathsf{Setup}(1^\lambda); \\
&\qquad (sk_i, pk_i) \xleftarrow{\mathrm{R}} \mathsf{Register}(\mathsf{param}, u_i), \qquad i = 1, \ldots, t; \\
&\qquad gpk \leftarrow \mathsf{Make\text{-}GPK}(\mathsf{param}, \{pk_1, \ldots, pk_t\}); \\
&\qquad gsk \leftarrow \mathsf{Make\text{-}GSK}(\mathsf{param}, \{pk_2, \ldots, pk_t\}, (sk_1, pk_1)) \quad | \\
&\qquad \mathsf{Anon\text{-}ID}^{\mathsf{V}}(\mathsf{param}, gpk)_{\mathsf{Anon\text{-}ID}^{\mathsf{P}}(\mathsf{param}, gpk, gsk)} = 1] \geq 1 - \mathsf{negl}(\lambda)
\end{aligned}
$$

4.3.2 Soundness

The Attack Game. We formalize the soundness guarantees that we require from an *Ad Hoc* Anonymous Identification scheme in terms of a game being played between an honest dealer and an adversary $\mathcal{A}$. In this game, the adversary is allowed to interact with three oracles $\mathcal{O}_{\mathsf{HReg}}$ (the *honest user registration oracle*), $\mathcal{O}_{\mathsf{Cor}}$ (the *user corruption oracle*), and $\mathcal{O}_{\mathsf{Scr}}$ (the *transcript oracle*) (*cf.* Figure 4.1).

The game begins with the honest dealer running the Setup algorithm for the security parameter 1^λ, and handing the resulting global parameters param to the adversary. Then, $\mathcal{A}$ arbitrarily interleaves queries to the three oracles, according to any adaptive strategy she wishes: eventually, she outputs a *target group* $S^* \subseteq \mathcal{PK}'$. At this point, $\mathcal{A}$ starts executing, in the role of the prover, a run of the $\mathsf{Anon\text{-}ID}$ protocol with the honest dealer, on common inputs param and $gpk^* \doteq \mathsf{Make\text{-}GPK}(\mathsf{param}, S^*)$. Notice that during such interaction, the adversary is still allowed to query the three oracles $\mathcal{O}_{\mathsf{HReg}}, \mathcal{O}_{\mathsf{Scr}}$ and $\mathcal{O}_{\mathsf{Cor}}$. Let $\tilde{\pi}$

Honest user registration oracle $\mathcal{O}_{\mathsf{HReg}}$		User corruption oracle $\mathcal{O}_{\mathsf{Cor}}$	
IN:	$u \in \mathcal{U}$	IN:	$pk_u \in \mathcal{PK}'$
RUN:	1. $(sk_u, pk_u) \xleftarrow{\mathrm{R}} \mathsf{Register}(\mathsf{param}, u)$	RUN:	1. $sk_u \leftarrow \mathsf{DB.Lookup}(pk_u)$
	2. $\mathsf{DB.Store}(sk_u, pk_u)$		/* $sk_u \leftarrow \perp$ if no match found */
OUT:	pk_u	OUT:	sk_u

Transcript oracle $\mathcal{O}_{\mathsf{Scr}}$	
IN:	$S' \subseteq \mathcal{PK}', pk_u \in \mathcal{PK}'$
RUN:	1. $sk_u \leftarrow \mathsf{DB.Lookup}(pk_u)$
	2. `if` $sk_u = \perp$
	3. `then` $\pi \leftarrow \perp$
	4. `else` $gpk \leftarrow \mathsf{Make\text{-}GPK}(\mathsf{param}, S' \cup \{pk_u\})$
	5. $gsk \leftarrow \mathsf{Make\text{-}GSK}(\mathsf{param}, S', (sk_u, pk_u))$
	6. $\pi \xleftarrow{\mathrm{R}} (\mathsf{Anon\text{-}ID}^{\mathsf{P}}(\mathsf{param}, gpk, gsk) \leftrightarrow \mathsf{Anon\text{-}ID}^{\mathsf{V}}(\mathsf{param}, gpk))$
OUT:	π

Figure 4.1: Oracles for the soundness attack game. DB denotes a database storing user secret key/public key pairs, indexed by public key.

be the transcript resulting from such run of the Anon-ID protocol. $\mathcal{A}$ wins the game if the following conditions hold:

1. for all $pk^* \in \mathcal{S}^*$, there is an entry indexed by pk^* in the SK-DB Database, **and**

2. $\tilde{\pi}$ is a valid transcript *i.e.*, the run completed with the honest dealer outputting 1, **and**

3. for all $pk^* \in \mathcal{S}^*$, $\mathcal{A}$ never queried $\mathcal{O}_{\mathsf{Cor}}$ on input pk^*;

Define $\mathsf{Succ}^{\mathsf{Snd}}_{\mathcal{A}}(\lambda)$ to be the probability that $\mathcal{A}$ wins the above game.

Definition 9. *An* Ad Hoc *Anonymous Identification scheme is sound against*

passive chosen-group attacks if any adversary $\mathcal{A}$ has negligible advantage to win the above game:

$$(\forall \lambda \in \mathbb{N})(\forall PPT\mathcal{A})[\mathsf{Succ}_{\mathcal{A}}^{\mathsf{Snd}}(\lambda) \leq \mathsf{negl}(\lambda)]$$

A Note on Active Security. Our definition of soundness models an adversary that, in her attempt to fool an honest verifier into accepting a "fake" run of the Anon-ID protocol, can actively (and, in fact, adaptively) corrupt users, but can only passively eavesdrop the communication between honest provers and verifiers. One could, of course, define stronger notions of security by considering active, concurrent or even reset attacks, along the lines of previous work on Identification Schemes [40, 5]; however, we refrain from doing so, both to keep the presentation simpler, and because the main application of our *Ad Hoc* Anonymous Identification schemes is to obtain new ring and group signatures scheme by means of the Fiat-Shamir Heuristic (see Section 4.6.3), for which security against a passive adversary suffices.

4.3.3 Anonymity

The Attack Game. We formalize the anonymity guarantees that we require from an *Ad Hoc* Anonymous Identification scheme in terms of a game being played between an honest dealer and an adversary $\mathcal{A}$. In this game, the adversary is allowed to interact only once with a "challenge" oracle $\mathcal{O}_{\mathsf{Ch}}$, described in Figure 4.2.

The game begins with the honest dealer running the Setup algorithm for the security parameter 1^{λ}, and handing the resulting global parameters param to the adversary. Then, the adversary $\mathcal{A}$ creates as many user secret key/public key pairs as she wishes, and experiments with the Make-GPK, Make-GSK, Anon-ID$^{\mathsf{P}}$

Challenge oracle $\mathcal{O}_{\mathsf{Ch}}$	
IN:	$S' \subseteq \mathcal{PK}', (sk_0, pk_0), (sk_1, pk_1)$
RUN:	1. $b^* \xleftarrow{\mathrm{R}} \{0,1\}$
	2. `if` $sk_0 \neq pk_0$ `or` $sk_1 \neq pk_1$ `then abort`
	3. $gpk \leftarrow \mathsf{Make\text{-}GSK}(\mathsf{param}, S' \cup \{pk_0, pk_1\})$
	4. $gsk^* \leftarrow \mathsf{Make\text{-}GSK}(\mathsf{param}, S' \cup \{pk_{1-b^*}\}, (sk_{b^*}, pk_{b^*}))$
	5. $\pi^* \xleftarrow{\mathrm{R}} (\mathsf{Anon\text{-}ID}^{\mathsf{P}}(\mathsf{param}, gpk, gsk^*) \leftrightarrow \mathsf{Anon\text{-}ID}^{\mathsf{V}}(\mathsf{param}, gpk))$
OUT:	π^*

Figure 4.2: The oracle for the anonymity attack game.

and $\mathsf{Anon\text{-}ID}^{\mathsf{V}}$ algorithms as long as she deems necessary; eventually, she queries the $\mathcal{O}_{\mathsf{Ch}}$ oracle, getting back a "challenge" transcript π^*. The adversary then continues experimenting with the algorithms of the system, trying to infer the random bit b^* used by the oracle $\mathcal{O}_{\mathsf{Ch}}$ to construct the challenge π^*; finally, $\mathcal{A}$ outputs a single bit $\tilde{b}$, her best guess to the "challenge" bit b^*.

Define $\mathsf{Succ}_{\mathcal{A}}^{\mathsf{Anon}}(\lambda)$ to be the probability that the bit $\tilde{b}$ output by $\mathcal{A}$ at the end of the above game is equal to the random bit b^* used by the $\mathcal{O}_{\mathsf{Ch}}$ oracle.

Definition 10. *An* Ad Hoc *Anonymous Identification scheme is* fully anonymizing *if any probabilistic, polynomial-time adversary $\mathcal{A}$ has success probability at most negligibly greater than one half:*

$$(\forall \lambda \in \mathbb{N})(\forall PPT\mathcal{A})\Big[\Big|\mathsf{Succ}_{\mathcal{A}}^{\mathcal{A}non}(\lambda) - \frac{1}{2}\Big| \leq \mathsf{negl}(\lambda)\Big]$$

4.3.4 Extension: Identity Escrow

In some scenarios, complete anonymity might create more security concerns than what it actually solves. Instead, some degree of "limited anonymity", not hindering user accountability, would be preferable. In our context, this can be

achieved with the help of a trusted *Identity Escrow Authority*, or IEA (also called *Anonymity Revocation Manager* elsewhere [25]), endowed with the capability of "reading" the identity of the prover "between the lines" of any transcript produced by a run of the Anon-ID protocol.

To enable such escrow capability, the definition of *Ad Hoc* Anonymous Identification scheme from Section 4.3.1 is modified as follows:

- The Setup algorithm is run by the IEA, and it additionally outputs an identity escrow key sk_{IE} (from some domain $\mathcal{SK}_{\mathsf{IE}}$), which the IEA keeps for himself.

- The Register algorithm is replaced by an efficient two-party protocol $(\mathsf{Register}^{\mathsf{user}}, \mathsf{Register}^{\mathsf{IEA}})$, meant to be run between the prospective user and the IEA, at the end of which the IEA learns the user's newly generated public key pk_u (possibly along with some other information aux_u about u that the IEA stores in a public registry database DB), but he doesn't learn anything about the corresponding secret key sk_u.

- The Anon-ID protocol is changed in that both prover and verifier get an additional common input called an *Identity Escrow Token* (IET) τ. A new IET τ is produced afresh by the prover before each execution of the Anon-ID protocol, and it should include a policy (or *label*) describing the circumstances under which the Identity Escrow Authority should honor an escrow request from a verifier.

- An additional (deterministic) Extract algorithm is defined, which takes as input an Identity Escrow Token τ and a transcript π for the Anon-ID protocol, along with the Identity Escrow secret key sk_{IE} and the registry

database DB, and returns a public key $pk \in \mathcal{PK}'$ or the special symbol $\perp$. Intuitively, $\mathsf{Extract}$ should be able to recover the identity of the user who participated as the prover in the run of the $\mathsf{Anon\text{-}ID}$ protocol that produced π as transcript; the symbol $\perp$ should be output when π does not meet the escrow criteria specified by the label included in τ, or when π is ill-formed (*e.g.*, when π comes from a ZK simulator, or upon failure to trace the correct identity).

Our definitions of the security properties of the system have to be adjusted, since we now have an additional functionality that the adversary may try to attack; moreover, the presence of the IEA may open new attack possibilities to the adversary.

The security requirements for the new $\mathsf{Extract}$ algorithm are formalized by augmenting the attack scenario defining the soundness property (Section 4.3.2). In this new, combined game, the adversary initially gets the IEA's secret key sk_{IE}, along with the public parameters param of the system. Also, the definition of the honest user registration oracle $\mathcal{O}_{\mathsf{HReg}}$ should be changed so as to use the two-party protocol $(\mathsf{Register}^{\mathsf{user}}, \mathsf{Register}^{\mathsf{IEA}})$ in place of the $\mathsf{Register}$ algorithm; similarly, the definition of the transcript oracle $\mathcal{O}_{\mathsf{Scr}}$ should be amended to reflect the syntactical changes to the $\mathsf{Anon\text{-}ID}$ protocol described above.

Then, the game proceeds as described in Section 4.3.2, except that we loosen the conditions under which the adversary is considered to win the game, substituting the last caveat with the following:

3′. for all $pk^* \in \mathcal{S}^*$, either $\mathsf{Extract}(\tilde{\pi}, sk_{\mathsf{IE}}, \mathsf{DB}) \neq pk^*$, **or** $\mathcal{A}$ never queried $\mathcal{O}_{\mathsf{Cor}}$ on input pk^*;

As for the anonymity property, the definition from Section 4.3.3 is changed

in that the adversary is now given access to two more oracles: a *corrupted-user registration oracle* $\mathcal{O}_{\mathsf{CReg}}() \doteq \mathsf{Register}^{\mathsf{IEA}}(sk_{\mathsf{IE}}, \mathsf{param}, \mathsf{DB})$, and a *user identity extraction oracle* $\mathcal{O}_{\mathsf{Xtr}}(\cdot,\cdot) \doteq \mathsf{Extract}(\cdot,\cdot,sk_{\mathsf{IE}}, \mathsf{DB})$. Also, the definition of the challenge oracle $\mathcal{O}_{\mathsf{Ch}}$ should be amended to reflect the syntactical changes to the Anon-ID protocol described above. (In particular, the challenge output by $\mathcal{O}_{\mathsf{Ch}}$ now consists of an IET τ^* and an associated transcript π^*.)

The adversary wins the game if she successfully guesses the random bit chosen by the challenge oracle $\mathcal{O}_{\mathsf{Ch}}$, without ever submitting to the extraction oracle $\mathcal{O}_{\mathsf{Xtr}}$ the IET τ^* which was output (together with the transcript π^*) by the challenge oracle $\mathcal{O}_{\mathsf{Ch}}$.

4.3.5 Extension: Supporting Incremental Group Growth

In many applications where *Ad Hoc* Anonymous Identification schemes could be useful, new *ad hoc* groups are often created as supersets of existing ones: for example, if *ad hoc* groups are used to enforce access control, new users may be added to the group of principals authorized to access a given resource. In such cases, the ability to "augment" a group public key with the a new user's public key can be very handy, especially if coupled with algorithms to efficiently create the corresponding group secret key for the new user, and to update the group secret keys for the existing users. Our model can be easily amended to capture this *incremental* functionality:

Definition 11. *An* incremental Ad Hoc *Anonymous Identification scheme is a* Ad Hoc *Anonymous Identification scheme augmented with 3 deterministic,*

polynomial-time algorithms

$$\textit{Augment-GPK} : \textsc{Par} \times \mathcal{GPK} \times \mathcal{P}(\mathcal{PK}) \rightarrow \mathcal{GPK}$$

$$\textit{Augment-Old-GSK} : \textsc{Par} \times \mathcal{GSK} \times \mathcal{P}(\mathcal{PK}) \rightarrow \mathcal{GSK}$$

$$\textit{Augment-New-GSK} : \textsc{Par} \times \mathcal{GPK} \times \mathcal{SK} \times \mathcal{PK} \rightarrow \mathcal{GSK}$$

such that:

- *(*Public Keys can be Incrementally Added into Group Public Keys*)*

$$\begin{aligned}
&(\forall \lambda \in \mathbb{N})(\forall (u'_1, \ldots, u'_{t'}) \in \mathcal{U}^*)(\forall (u''_1, \ldots, u''_{t''}) \in \mathcal{U}^*) \\
&\quad Pr[\textit{param} \xleftarrow{R} \textit{Setup}(1^\lambda); \\
&\qquad (sk'_i, pk'_i) \xleftarrow{R} \textit{Register}(\textit{param}, u'_i), \qquad i = 1, \ldots, t'; \\
&\qquad (sk''_i, pk''_i) \xleftarrow{R} \textit{Register}(\textit{param}, u''_i), \qquad i = 1, \ldots, t''; \\
&\qquad gpk' \leftarrow \textit{Make-GPK}(\textit{param}, \{pk'_1, \ldots, pk'_{t'}\}); \\
&\qquad gpk \leftarrow \textit{Make-GPK}(\textit{param}, \{pk'_1, \ldots, pk'_{t'}, pk''_1, \ldots, pk''_{t''}\}); \\
&\qquad \widehat{gpk} \leftarrow \textit{Augment-GPK}(\textit{param}, gpk', \{pk''_1, \ldots, pk''_{t''}\}) \quad | \\
&\qquad \widehat{gpk} = gpk] \geq 1 - \mathsf{negl}(\lambda)
\end{aligned}$$

- *(*Public Keys can be Incrementally Added into Group Secret Keys*)*

$$
\begin{aligned}
&(\forall \lambda \in \mathbb{N})(\forall (u'_1, \ldots, u'_{t'}) \in \mathcal{U}^*)(\forall (u''_1, \ldots, u''_{t''}) \in \mathcal{U}^*) \\
&\quad Pr[\mathit{param} \stackrel{R}{\leftarrow} \mathit{Setup}(1^\lambda); \\
&\qquad (sk'_i, pk'_i) \stackrel{R}{\leftarrow} \mathit{Register}(\mathit{param}, u'_i), \qquad i = 1, \ldots, t'; \\
&\qquad (sk''_i, pk''_i) \stackrel{R}{\leftarrow} \mathit{Register}(\mathit{param}, u''_i), \qquad i = 1, \ldots, t''; \\
&\qquad gsk' \leftarrow \mathit{Make\text{-}GSK}(\mathit{param}, \{pk'_2, \ldots, pk'_{t'}\}, (sk'_1, pk'_1)); \\
&\qquad gsk \leftarrow \mathit{Make\text{-}GSK}(\mathit{param}, \{pk'_2, \ldots, pk'_{t'}, pk''_1, \ldots, pk''_{t''}\}, (sk'_1, pk'_1)); \\
&\qquad \widehat{gsk} \leftarrow \mathit{Augment\text{-}Old\text{-}GSK}(\mathit{param}, gsk', \{pk''_1, \ldots, pk''_{t''}\}) \quad | \\
&\qquad \widehat{gsk} = gsk] \geq 1 - \mathsf{negl}(\lambda)
\end{aligned}
$$

- *(*Group Secret Keys can be Incrementally Built from Group Public Keys*)*

$$
\begin{aligned}
&(\forall \lambda \in \mathbb{N})(\forall (u_1, \ldots, u_t) \in \mathcal{U}^*) \\
&\quad Pr[\mathit{param} \stackrel{R}{\leftarrow} \mathit{Setup}(1^\lambda); \\
&\qquad (sk_i, pk_i) \stackrel{R}{\leftarrow} \mathit{Register}(\mathit{param}, u_i), \qquad i = 1, \ldots, t; \\
&\qquad gpk \leftarrow \mathit{Make\text{-}GPK}(\mathit{param}, \{pk_2, \ldots, pk_t\}); \\
&\qquad gsk \leftarrow \mathit{Make\text{-}GSK}(\mathit{param}, \{pk_2, \ldots, pk_t\}, (sk_1, pk_1)); \\
&\qquad \widehat{gsk} \leftarrow \mathit{Augment\text{-}New\text{-}GSK}(\mathit{param}, gpk, (sk_1, pk_1)) \quad | \\
&\qquad \widehat{gsk} = gsk] \geq 1 - \mathsf{negl}(\lambda)
\end{aligned}
$$

where all the probabilities are over the random coins of the Setup *and* Register *algorithms.*

4.4 Generic Construction

In this section, we will establish the fact that the existence of accumulators with one way domain implies the existence of *Ad Hoc* Anonymous Identification schemes. Below we describe how the algorithms (Setup, Register, Make-GPK, Make-GSK, Anon-ID$^{\mathsf{P}}$, Anon-ID$^{\mathsf{V}}$) can be implemented given an accumulator with one-way domain $(\{F_\lambda\}_{\lambda\in\mathbb{N}}, \{X_\lambda\}_{\lambda\in\mathbb{N}}, \{Z_\lambda\}_{\lambda\in\mathbb{N}}, \{R_\lambda\}_{\lambda\in\mathbb{N}},)$.

- Setup executes the accumulator generation algorithm G on 1^λ to obtain $f \in F_\lambda$. Then it samples U_f to obtain $u \in_R U_f$. Setup terminates by setting param $:= (\lambda, u, f, D, W)$, where D and W are polynomial-time algorithms respectively to decide and to sample the relation R_λ.

- Register first samples a pair $(x, z) \in X_\lambda \times Z_\lambda$ such that $(x, z) \in R_\lambda$ using the sampling algorithm W of the relation R_λ on input 1^λ. Then, Register outputs $sk \doteq z$ (the user secret key) and $pk \doteq x$ (the user public key). Observe that $\mathcal{SK}' = \mathcal{SK} \doteq Z_\lambda$, $\mathcal{PK}' = X_f^{\mathsf{ext}}$ and $\mathcal{PK} \doteq X_\lambda$.

- Make-GPK operates as follows: given a set of user public keys $S = \{x_1, \ldots, x_t\}$ and the parameters (λ, u, f, D), it sets the group public key of S to be the (unique) accumulated value of S over u *i.e.*, $gpk_S \doteq f(u, S)$. Note that thanks to the quasi-commutativity property of f, Make-GPK is indeed order-independent.

- Make-GSK operates as follows: given the set of user public keys $S' \doteq \{x_1, \ldots, x_t\}$, a user secret key/public key pair (z, x) and the system parameters param $= (\lambda, u, f, D, W)$, it first computes the accumulated value $w \doteq f(u, S')$, and then sets the group secret key gsk to be the tuple

(x, z, w). Observe that w is a witness for x in $f(u, S)$ (where $S \doteq S' \cup \{x\}$), and that $\mathcal{GSK} \doteq X_\lambda \times Z_\lambda \times U_f$ and $\mathcal{GPK} \doteq U_f$.

- $\mathsf{Anon\text{-}ID}^{\mathsf{P}}$ and $\mathsf{Anon\text{-}ID}^{\mathsf{V}}$ are obtained generically as the Σ-protocol corresponding to the following **NP**-relation $\mathcal{R}_{\mathsf{param}} \subset \mathcal{GPK} \times \mathcal{GSK}$:

$$\mathcal{R}_{\mathsf{param}} \doteq \big\{ \big(v, (x, z, w)\big) \mid \big((x, z) \in R_\lambda\big) \wedge \big(f(w, x) = v\big) \big\}$$

  It is easy to see that the above relation is polynomial-time verifiable: indeed, given v and (x, z, w), one can check in time polynomial in $|v|$ whether $(x, z) \in R_\lambda$ (by verifying that $D(x, z) = 1$), and whether w is indeed a witness for x in v (by verifying that $f(w, x) = v$). Thus, by Theorem 6, we can construct a Σ-protocol (P, V) for the **NP**-relation $\mathcal{R}_{\mathsf{param}}$. In the resulting protocol, the common input to the prover and the verifier is the accumulated value v (*i.e.*, a group public key) and the additional input to the prover is a tuple of the form (x, z, w) (*i.e.*, a group secret key). Hence, the protocol (P, V) meets the specification of the $\mathsf{Anon\text{-}ID}$ protocol.

Correctness of the above construction follows from the fact that relation $\mathcal{R}_{\mathsf{param}}$ is essentially equivalent to the $\rightleftharpoons$ relation. Consequently, a prover holding a group secret key $gsk \doteq (x, z, w)$ $\rightleftharpoons$-related to the group public key $gpk \doteq v$ given as input to the verifier, possesses a tuple belonging to the relation $\mathcal{R}_{\mathsf{param}}$, so that the execution of the $\mathsf{Anon\text{-}ID}$ protocol will terminate with the verifier outputting 1, with overwhelming probability.

We also remark that, thanks to the quasi-commutativity of (one-way) accumulators, our construction can be extended to meet the extra requirements of

"incremental" *Ad Hoc* Anonymous Identification scheme (*cf.* Section 4.3.5) in a straightforward and efficient way.

4.4.1 Soundness

Intuitively, the soundness of the above generic construction stems from the following considerations. The Special Honest-Verifier Zero-Knowledge property of the Σ-protocol $\mathsf{Anon\text{-}ID}$ guarantees that the Transcript Oracle doesn't leak any information to the adversary that she could not compute herself. By the Special Soundness property, in order to make the honest dealer accept (with non-negligible probability) a run of the $\mathsf{Anon\text{-}ID}$ protocol in which the group public key $gpk^* \doteq v^*$ consists solely of the aggregation of public keys of non-corrupted users, $\mathcal{A}$ should posses a tuple $gsk \doteq (x^*, z^*, w^*)$ such that $(x^*, z^*) \in R_\lambda$ and w^* is a witness of x^* in v^*. Now, the collision resistance of the accumulator implies that the user public key x^* must indeed have been accumulated within v^*, which means (by the third caveat of the soundness attack game in Section 4.3.2) that x^* belongs to a non-corrupted user. Hence, the adversary didn't obtain the pre-image z^* via the user corruption oracle, which implies that $\mathcal{A}$ was able to find it by herself, contradicting the one-wayness of the accumulator's domain.

More formally, we now present a reduction argument showing how an adversary $\mathcal{A}$ having non-negligible advantage $\mathsf{Succ}^{\mathsf{Snd}}_{\mathcal{A}}(\lambda)$ in attacking the soundness of the above scheme can be used to attack the security of the underlying accumulator with one-way domain. Recall from Section 4.2.3 that a secure accumulator with one-way domain is such that:

1. the advantage $\mathsf{Succ}^{\mathsf{Coll}}_{\mathcal{B}_1}(\lambda)$ of any probabilistic, polynomial-time adversary $\mathcal{B}_1$ in attacking the collision-resistance of the accumulator is a negligible

function in λ;

2. the advantage $\mathsf{Succ}^{\mathsf{OW}}_{\mathcal{B}_2}(\lambda)$ of any probabilistic, polynomial-time adversary $\mathcal{B}_2$ in attacking the one-wayness of the accumulator's domain is a negligible function in λ.

Thus, our reduction argument will proceed as follows: given black-box access to $\mathcal{A}$, we will construct two adversaries, $\mathcal{B}_1$ and $\mathcal{B}_2$, attacking the security of the accumulator with one-way domain in the sense of 1 and 2 above, respectively. We will then prove that if $\mathsf{Succ}^{\mathsf{Snd}}_{\mathcal{A}}(\lambda)$ is non-negligible, then either $\mathsf{Succ}^{\mathsf{Coll}}_{\mathcal{B}_1}(\lambda)$ or $\mathsf{Succ}^{\mathsf{OW}}_{\mathcal{B}_2}(\lambda)$ (or both) must also be non-negligible, thus reaching a contradiction.

Construction of B_1. We now describe how to turn $\mathcal{A}$ into an adversary $\mathcal{B}_1$ attacking the collision-resistance of the accumulator with one-way domain. The input to $\mathcal{B}_1$ is the description of the accumulator function f, the initial value u, and the decision and sampling algorithms D and W for the relation R_λ. This is exactly the format of the system parameters param that $\mathcal{A}$ expects, so $\mathcal{B}_1$ can feed $\mathcal{A}$ with that. During its execution, $\mathcal{A}$ also expects to be given access to the three oracles $\mathcal{O}_{\mathsf{HReg}}$, $\mathcal{O}_{\mathsf{Cor}}$ and $\mathcal{O}_{\mathsf{Scr}}$: $\mathcal{B}_1$ simply simulates these oracles honestly, according to the specification in Figure 4.1 (in particular, $\mathcal{B}_1$ will use the sampling algorithm W to generate user secret key/public key pairs). Eventually, $\mathcal{A}$ will tell $\mathcal{B}_1$ the ad hoc group S^* that she wishes to target, and then she will initiate a run of the $\mathsf{Anon\text{-}ID}$ protocol on the common input $gpk^* \doteq f(u, S^*)$: let COM^* be the first flow sent by $\mathcal{A}$ to $\mathcal{B}_1$. Adversary $\mathcal{B}_1$ then attempts, using standard rewinding techniques for Σ-protocols [72], to extract a tuple (x^*, z^*, w^*) from $\mathcal{A}$: if successful, $\mathcal{B}_1$ outputs (x^*, w^*, S^*), otherwise $\mathcal{B}_1$ aborts.

Construction of B_2. The construction of adversary $\mathcal{B}_2$ follows the same struc-

ture as the one given above for $\mathcal{B}_1$, with few differences (described below), stemming from the fact that the input to $\mathcal{B}_2$ additionally includes a "challenge" value $\hat{x}$ for which $\mathcal{B}_2$ needs to find a preimage $\hat{z}$.

Let Q_{HReg} be an estimate on the number of queries that $\mathcal{A}$ will ask to $\mathcal{O}_{\mathsf{HReg}}$, and let i be a random integer between 1 and Q_{HReg}; the simulation put on by $\mathcal{B}_2$ will proceed as done by $\mathcal{B}_1$, with the following changes:

- In answering the i^{th} query to oracle $\mathcal{O}_{\mathsf{HReg}}$, $\mathcal{B}_2$ will return the challenge value $\hat{x}$ to $\mathcal{A}$;
- If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Cor}}$ on $\hat{x}$, $\mathcal{B}_2$ immediately aborts the simulation;
- If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Scr}}$ on $(S', \hat{x})$, $\mathcal{B}_2$ uses the simulator for the Σ-protocol to produce the transcript π, and returns it to $\mathcal{A}$;
- After successfully rewinding $\mathcal{A}$ and extracting (x^*, z^*, w^*), $\mathcal{B}_2$ checks if $x^* = \hat{x}$: if so, $\mathcal{B}_2$ outputs z^*; if not, $\mathcal{B}_2$ aborts.

Reaching a contradiction. Consider the events:

$$E_{\mathsf{rew}} \doteq \text{“rewinding succeeded”} \qquad E_{\mathsf{in}} \doteq \text{“}x^* \in S^*\text{”} \qquad E_{\mathsf{eq}} \doteq \text{“}x^* = \hat{x}\text{”},$$

and the quantities:

$$p_{\mathsf{rew}} \doteq \Pr[E_{\mathsf{rew}}] \qquad p_1 \doteq \Pr[\overline{E_{\mathsf{in}}} \mid E_{\mathsf{rew}}] \qquad p_2 \doteq \Pr[E_{\mathsf{eq}} \mid E_{=}sfrew \wedge E_{\mathsf{in}}],$$

the probability being over the random coins of $\mathcal{A}$, $\mathcal{B}_1$, $\mathcal{B}_2$ and the random choice of i. Now, notice that if during the execution of adversary $\mathcal{B}_1$, events E_{rew} and $\overline{E_{\mathsf{in}}}$ occur, then the output of $\mathcal{B}_1$ is actually a collision, so that $\mathcal{B}_1$ wins the game: it follows that $\mathsf{Succ}^{\mathsf{Coll}}_{\mathcal{B}_1}(\lambda) \geq p_{\mathsf{rew}} \cdot p_1$. As for executions of adversary $\mathcal{B}_2$, first notice that the simulation is correct as long as $\mathcal{A}$ does not query $\mathcal{O}_{\mathsf{Cor}}$ on

$\hat{x}$ (in particular, Special Honest-Verifier Zero-Knowledge implies that $\mathcal{A}$ cannot detect that transcripts from queries of the form $\mathcal{O}_{\mathsf{Scr}}(S', \hat{x})$ actually come from the Simulator). Thus, if events E_{rew}, E_{in} and E_{eq} occur, then the output of $\mathcal{B}_2$ is actually a pre-image of $\hat{x}$, so that $\mathcal{B}_2$ wins the game: it follows that $\mathsf{Succ}^{\mathsf{OW}}_{\mathcal{B}_2}(\lambda) \geq p_{\mathsf{rew}} \cdot (1 - p_1) \cdot p_2$. The probability p_{rew} of successful rewinding can be safely estimated (*cf.* [72]) as $p_{\mathsf{Rew}} = O(\mathsf{Succ}^{\mathsf{Snd}}_{\mathcal{A}}(\lambda)^2)$. As for p_2, notice that since the choice of i is independent from $\mathcal{A}$'s view, it follows that p_2 is at least $1/Q_{\mathsf{HReg}}$, which is non-negligible. Finally, observe that p_1 and $(1 - p_1)$ cannot both be negligible, so that, assuming that $\mathsf{Succ}^{\mathsf{Snd}}_{\mathcal{A}}(\lambda)$ is non-negligible, it follows that either $\mathsf{Succ}^{\mathsf{Coll}}_{\mathcal{B}_1}(\lambda)$ or $\mathsf{Succ}^{\mathsf{OW}}_{\mathcal{B}_2}(\lambda)$ (or both) are also non-negligible, as required. □

4.4.2 Anonymity

In attacking the anonymity of the proposed scheme, the adversary basically chooses a group public key $gpk \doteq v$ and two group secret keys $gsk_1 \doteq (x_1, z_1, w_1)$ and $gsk_2 \doteq (x_2, z_2, w_2)$, both $\rightleftharpoons$-related to gpk. To subvert anonymity, the adversary should then be able (*cf.* Section 4.3.3) to tell whether gsk_1 or gsk_2 was used in producing the (honest) "challenge" transcript. Since in the generic construction above the Anon-ID protocol is implemented as a Σ-protocol, this would mean that the adversary is able to tell which "witness" (gsk_1 or gsk_2) was used by the prover to show that v belongs to the **NP**-language $\mathcal{L}_{\mathsf{param}}$ associated to the **NP**-relation $\mathcal{R}_{\mathsf{param}}$. In other words, a successful adversary would break the Witness Indistinguishability of the Anon-ID protocol, which contradicts the fact that Anon-ID enjoys Special Honest-Verifier Zero-Knowledge.

To turn the above intuition into a formal proof, we now define two games,

$\mathbf{G}_0$ and $\mathbf{G}_1$, indistinguishable to the eyes of any probabilistic, polynomial-time adversary, both defined over the same probability space, where $\mathbf{G}_0$ is the original anonymity attack game defined in Section 4.3.3, and $\mathbf{G}_1$ is a game in which even an unbounded adversary cannot win with probability better than 1/2.

Game $\mathbf{G}_0$. Define game $\mathbf{G}_0$ to be the original anonymity attack game (*cf.* Section 4.3.3).

Game $\mathbf{G}_1$. In game $\mathbf{G}_1$, step 4. and 5. of the Challenge Oracle $\mathcal{O}_{\mathsf{Ch}}$ from Figure 4.2 are replaced by the following:

$$4'. \quad \mathsf{Ch} \xleftarrow{\mathrm{R}} \{0,1\}^{\chi}$$

$$5'. \quad \pi^* \leftarrow \mathsf{Sim}(\mathsf{param}, gpk, \mathsf{Ch})$$

where χ is the challenge size and Sim is the simulator for the Σ-protocol.

In other words, in game $\mathbf{G}_1$, the Challenge Oracle constructs the challenge using the Simulator algorithm, so that the value of b^* is independent of the adversary's view. By Special Honest-Verifier Zero-Knowledge, no probabilistic, polynomial-time adversary can detect that the challenge transcript π^* actually comes from the Simulator, so that the probability of adversary $\mathcal{A}$ guessing b^* in $\mathbf{G}_0$ and $\mathbf{G}_1$ can only be negligibly apart. But as argued above, such probability is exactly 1/2 in $\mathbf{G}_1$: it follows that any probabilistic, polynomial-time adversary $\mathcal{A}$ can only guess b^* in the original anonymity attack scenario of Section 4.3.3 with probability at most negligibly greater than 1/2 *i.e.*, our generic construction yields a *fully anonymizing Ad Hoc* Anonymous Identification scheme.

4.4.3 Adding ID Escrow

The generic construction described above can be extended to deal with Identity Escrow as follows. During the initialization, the Setup algorithm additionally

runs the key generation algorithm $\mathcal{K}$ of some CCA2-secure encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$. The resulting public key pk_{IE} is included in the system parameters param, and the secret key sk_{IE} is given to the Identity Escrow Authority (IEA).

As for the user registration phase, each new user, after choosing his user secret key/public key pair $(sk, pk) \doteq (z, x)$, registers his public key with the IEA, which simply stores his identity and public key in a publicly-available database DB.

The Anon-ID protocol is also changed to be the Σ-protocol corresponding to the following **NP**-relation:

$$\mathcal{R}^{\mathsf{IE}}_{\mathsf{param}} \doteq \left\{ \big((v, \tau), (x, z, w, r)\big) \mid \big((x, z) \in R_\lambda\big) \wedge \big(f(w, x) = v\big) \wedge \big(\tau = \mathcal{E}_{pk_{\mathsf{IE}}}(x; r)\big) \right\}$$

In other words, the prover now additionally encrypts his public key x under the IEA's public key pk_{IE}, and proves to the verifier that he did so correctly. Notice that the Identity Escrow Token τ (*cf.* Section 4.3.4) is created by the prover and sent to the verifier immediately before each execution of the Anon-ID protocol.

Finally, on input an IET τ and a transcript π, the Extract algorithm, recovers the identity of the user that played the role of the prover by decrypting τ.

It is not hard to check that the above changes do not affect the soundness and anonymity properties of the generic construction: in particular, the CCA2-security of the encryption scheme (which is needed since a malicious party could trick the IEA into acting as a decryption oracle) guarantees that an honestly-created IET τ cannot be modified so as to alter the prover identity hidden within it. We omit the details.

4.4.4 Supporting Incremental Growth

Below we sketch how the algorithms for incremental *ad hoc* groups can be efficiently implemented for the generic construction of Section 4.4 (and also for its efficient instantiations of Section 4.5).

Recall that the systems parameters of an instance of our generic construction (*cf.* Section 4.4) have the form $\mathsf{param} := (\lambda, u, f, D, W)$. Also, recall that $\mathcal{SK} \doteq Z_\lambda$, $\mathcal{PK} \doteq X_\lambda$, $\mathcal{GSK} \doteq X_\lambda \times Z_\lambda \times U_f$ and $\mathcal{GPK} \doteq U_f$.

Then, the algorithms $\mathsf{Augment\text{-}GPK}, \mathsf{Augment\text{-}Old\text{-}GSK}, \mathsf{Augment\text{-}New\text{-}GSK}$ can be defined as follows:

$$\begin{aligned}
\mathsf{Augment\text{-}GPK}(\mathsf{param}, v, S'') &\doteq f(v, S'') \\
\mathsf{Augment\text{-}Old\text{-}GSK}(\mathsf{param}, (x, z, w), S'') &\doteq (x, z, f(w, S'')) \\
\mathsf{Augment\text{-}New\text{-}GSK}(\mathsf{param}, (x, v, z, x) &\doteq (x, z, f(v, x))
\end{aligned}$$

4.5 Efficient Implementation

4.5.1 Efficient Accumulators with One-way Domain

An efficient construction of a collision-resistant accumulator was presented in [25], based on earlier work by [4] and [12]. Based on this construction, we present an efficient implementation of an accumulator with one-way domain.

For $\lambda \in \mathbb{N}$, the family F_λ consists of the exponentiation functions modulo λ-bit rigid integers:

$$f : (\mathbb{Z}_n^*)^2 \times \mathbb{Z}_{n/4} \to (\mathbb{Z}_n^*)^2$$

$$f : (u, x) \mapsto u^x \bmod n$$

where $n \in \mathsf{Rig}_\lambda$ and $(\mathbb{Z}_n^*)^2$ denotes the set of quadratic residues modulo n.

The accumulator domain $\{X_\lambda\}_{\lambda\in\mathbb{N}}$ is defined by:

$$X_\lambda \doteq \left\{ e \text{ prime } \mid \left(\frac{e-1}{2} \in \mathsf{RSA}_\ell\right) \wedge \left(e \in S(2^\ell, 2^\mu)\right)\right\}$$

where $S(2^\ell, 2^\mu)$ is the integer range $(2^\ell - 2^\mu, 2^\ell + 2^\mu)$ that is embedded within $(0, 2^\lambda)$ with $\lambda - 2 > \ell$ and $\ell/2 > \mu + 1$. The pre-image domain $\{Z_\lambda\}_{\lambda\in\mathbb{N}}$ and the one-way relation $\{R_\lambda\}_{\lambda\in\mathbb{N}}$ are defined as follows:

$$Z_\lambda \doteq \{(e_1, e_2) \mid e_1, e_2 \text{ are distinct } \ell/2\text{-bit primes and } e_2 \in S(2^{\ell/2}, 2^\mu)\}$$

$$R_\lambda \doteq \{\left(x, (e_1, e_2)\right) \in X_\lambda \times Z_\lambda \mid \left(x = 2e_1e_2 + 1\right)\}$$

The collision resistance of the above construction can be based on the Strong RSA Assumption, as showed in [25]. Regarding the added one-wayness of the domain, assuming the hardness of factoring RSA integers, it is easy to see that the **NP**-relation R_λ satisfies our one-wayness requirement (*cf.* Section 4.2.3): hence, the above construction yields a secure accumulator with one-way domain.

4.5.2 Efficient Proof of Witnesses for the Accumulator

The generic construction described in Section 4.4 derives algorithms $\mathsf{Anon\text{-}ID}^\mathsf{P}$ and $\mathsf{Anon\text{-}ID}^\mathsf{V}$ from the Σ-protocol corresponding to some **NP**-relation $\mathcal{R}_{\mathsf{param}}$: for our RSA-based accumulator with one-way domain, the relation is defined as:

$$\mathcal{R}_{\mathsf{param}}^{RSA} \doteq \{\left(v, (x, (e_1, e_2), w)\right) \mid \left(w^x \equiv v \bmod n\right) \wedge \left(x \in S(2^\ell, 2^\mu)\right) \wedge \left(x - 1 = 2e_1e_2\right) \wedge \left(e_2 \in S(2^{\ell/2}, 2^\mu)\right)\}$$

However, the protocol generically obtained in virtue of Theorem 6, though polynomial time, is not efficient enough to be useful in practice; thus, below we

describe how a practical Σ-protocol for relation $\mathcal{R}^{RSA}_{\mathsf{param}}$ could be constructed, exploiting the framework of discrete-log relation sets [62], which provides a simple method to construct complex proofs of knowledge over groups of unknown order. A discrete-log relation set R is a set of vectors of length m defined over $\mathbb{Z} \cup \{\alpha_1, \ldots, \alpha_r\}$ (where the α_j's are called the free variables of the relation) and involves a sequence of base elements $A_1, \ldots, A_m \in (\mathbb{Z}^*_n)^2$. For any vector $\langle a^i_1, \ldots, a^i_m \rangle$ the corresponding relation is defined as $\prod_{j=1}^{m} A_i^{a^i_j} = 1$. The conjunction of all the relations is denoted as $R(\alpha_1, \ldots, \alpha_r)$. In [62], an efficient Σ-protocol is presented for any discrete-log relation set R, by which the prover can prove of knowledge of a sequence of witnesses $x_1, \ldots, x_r$, with $x_i \in S(2^{\ell_i}, 2^{\mu_i})$ that satisfy $R(x_1, \ldots, x_r) \wedge \Big(\wedge_{i=1}^{r} (x_i \in S(2^{\ell_i}, 2^{\epsilon(\mu_i+k)+2}) \Big)$, where $\epsilon > 1, k \in \mathbb{N}$ are security parameters. Note that the tightness of the integer ranges can be increased by employing the range proofs of [19], nevertheless the tightness achieved above is sufficient for our purposes, and incurs a lower overhead.

In order to prove the relation $\mathcal{R}^{RSA}_{\mathsf{param}}$, we assume that the public parameters param include the elements $g, h, y, t, s \in (\mathbb{Z}^*_n)^2$ with unknown relative discrete-logarithms. In order to construct the proof, the prover provides a sequence of public values T_1, T_2, T_3, T_4, T_5 such that $T_1 = g^r, T_2 = h^r g^x, T_3 = s^r g^{e_2}, T_4 = wy^r, T_5 = t^r g^{2e_1}$, where $r \xleftarrow{\mathrm{R}} [0, \lfloor n/4 \rfloor - 1]$.

The proof is constructed as a discrete-log relation set that corresponds to the equations $T_1 = g^r$, $T_2 = h^r g^x$, $(T_1)^x = g^{a_1}$, $(T_1)^{e_2} = g^{a_2}$, $T_3 = s^r g^{e_2}$ $(T_4)^x = v y^{a_1}$, $(T_5)^{e_2} g = t^{a_2} g^x$, for the free variables r, x, e_2, a_1, a_2 such that $x \in S(2^\ell, 2^\mu), e_2 \in S(2^{\ell/2}, 2^\mu)$, $a_1 = rx$ and $a_2 = re_2$. The matrix of the discrete-log relation set is shown below:

$$
\left[\begin{array}{rcccccccccccc}
 & g & h & y & t & s & v & T_1^{-1} & T_2^{-1} & T_3^{-1} & T_4^{-1} & T_5^{-1} & g^{-1} \\
T_1 = g^r : & r & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
T_2 = h^r g^x : & x & r & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
(T_1)^x = g^{a_1} : & a_1 & 0 & 0 & 0 & 0 & 0 & x & 0 & 0 & 0 & 0 & 0 \\
T_3 = s^r g^{e_2} : & e_2 & 0 & 0 & 0 & r & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
(T_1)^{e_2} = g^{a_2} : & a_2 & 0 & 0 & 0 & 0 & 0 & e_2 & 0 & 0 & 0 & 0 & 0 \\
(T_4)^x = v y^{a_1} : & 0 & 0 & a_1 & 0 & 0 & 1 & 0 & 0 & 0 & x & 0 & 0 \\
(T_5)^{e_2} g = t^{a_2} g^x : & x & 0 & 0 & a_2 & 0 & 0 & 0 & 0 & 0 & 0 & e_2 & 1
\end{array}\right]
$$

Observe that a proof of the above discrete-log relation set ensures that (i) the prover knows a witness w for some value x in the *ad hoc* group accumulated value v, and (ii) for the same x, the value $x - 1$ can be split by the prover into two integers one of which belongs to $S(2^{\ell/2}, 2^{\mu})$. This latter range-property guarantees the non-triviality of the splitting *i.e.*, that the prover knows a non-trivial factor of $x - 1$ (*i.e.*, different than $-1, 1, 2$). Note that this will require that the parameters ℓ, μ, ϵ, k should be selected such that $\ell/2 > \epsilon(\mu + k) + 2$.

4.5.3 ID Escrow

In Section 4.4.3, we discussed a generic transformation to add Identity Escrow to an *Ad Hoc* Anonymous Identification scheme. Most of the required changes do not affect the system's efficiency, except for the need to resort to a generic derivation of the Anon-ID protocol.

This performance penalty is not unavoidable, however: in fact, escrow capabilities can be directly supported by the Σ-protocol for Anonymous Identification described in Section 4.5.2. using protocols for verifiable encryption and

decryption of discrete logarithms from [27].

With notation as in Section 4.5.2, the Anon-ID protocol is augmented as follows: after sending the commitment T_2 to the verifier, the prover verifiably encrypts an opening of T_2 (namely, x and r) under the IEA public key. By checking that the encryption was carried out correctly, the verifier can be assured that, should the need arise, the IEA would be able to identify the prover by decrypting such opening, which would yield the prover's public key x. Moreover, by using verifiable decryption in the Extract algorithm, we can prevent the IEA from untruthfully opening the identity of the prover for a given transcript, or falsely claiming that the prover's identity cannot be properly recovered.

Alternatively, if only honest users are assumed to have access to the Escrow functionality (so that malicious parties cannot exploit the IEA as a "decryption oracle"), then a more efficient solution is possible, by having the IEA knowing the value $\log_g(h)$ in the proof of knowledge from Section 4.5. Then, given a transcript of the protocol (which includes the values T_1, T_2, T_3, T_4, T_5) the IEA can recover the value $g^x = T_2 T_1^{-\log_g(h)}$, from which the prover's identity can be recovered by comparing g^x to the public keys published in the public DB database.

4.6 Applications

4.6.1 *Ad Hoc* Identification Schemes

This is the most direct application. Imagine a large universe of users, where each user has a public certificate, but otherwise there is no central authority in the system. Now, some party "from the street" has a resource which he

is willing to share with some subset of the users. For example, an Internet provider P may want to enable internet access to all its subscribers. However, privacy considerations may lead a user to refuse to positively identify himself; in fact, this is not strictly necessary, as long as he proves he belongs to the group of current subscribers. Our *ad hoc* identification schemes are ideally suited for this application, bringing several very convenient feautures. First, P can simply take all the public keys of the users (call this set S) and combine them into one short group public key gpk_S. Notice, this initial setup is the only operation P performs which requires time proportional to the group size. As for each user $u \in S$, once again he will use his secret key and the public keys of other user to prepare one short group secret key gsk_u. After that, all identifications that u makes to P require computation and communication independent of the size of the group. Now, another provider P' can do the same for a totally different subgroup, and so on, allowing truly *ad hoc* groups with no trusted authority needed in the system. Additionally, with incremental *Ad Hoc* Anonymous Identification schemes (*cf.* Section 4.3.5), one can preserve efficiency even when the *ad hoc* group is built gradually, as each new member addition only requires constant computation by P and by every pre-existing user in the system.

4.6.2 Constant Size Ring Signatures

This is one of our main applications, since it dramatically improves the efficiency of all known ring signature schemes (*e.g.*, [73, 22, 18]). Recall, in a ring signature scheme there again is a universe of registered users, but no trusted authority. Any user u can then form a ring S, and sign a message m in such a way that any verifier (who knows S) can confidently conclude that "the message m

was signed by some member u of S", but gets no information about u beyond $u \in S$. Previous papers on the subject suggested that linear dependence of the ring signature size on the size of the ring S was inevitable, since the group is *ad hoc*, so the verifier needs to know at least the description of the ring. While the latter is true, in practical situations the ring often stays the same for a long period of time (in fact, there could be many "popular" rings that are used very often by various members of the ring), or has an implicit short decryption (*e.g.*, the ring of public keys of all members of the President's Cabinet). Thus, we feel that the right measure of "signature size" in this situation is that of an "actual signature"—the string one needs in addition to the group description. Indeed, when the ring stays the same for a long time or has a short description, this actual signature is all that the verifier needs in order to verify its correctness. With this in mind, there is no reason why the signature size must be linear in the size of the ring.

In fact, our result shows that it does not have to be. Specifically, by applying the Fiat-Shamir heuristics to our *ad hoc* identification scheme, we immediately get ring signatures of constant size. Moreover, our ring signatures enjoy several desirable features not generally required by ring signatures (even those of constant size). For example, both the signer and the verifier need to perform a one-time computation proportional to the size of the ring, and get some constant-size information (gsk_S and gpk_S, respectively) which allows them to produce/verify many subsequent signatures in constant time.

4.6.3 *Ad Hoc* ID Escrow and Group Signatures

As mentioned in Section 4.3.4, in some situations complete anonymity might not be desirable. In this case, one wishes to introduce a trusted *Identity Escrow Authority* (IEA), who can reveal the true identity of the user given the transcript of the identification procedure (presumably, when some "anonymity abuse" happens). Such schemes are called ID Escrow schemes [63] and have traditionally been considered for fixed groups. ID Escrow schemes are duals of group signature schemes [29, 1], which again maintain a single group of signers, and where a similar concern is an issue when signing a document anonymously. As argued in Section 4.4.3 and Section 4.5.3, our *Ad Hoc* Anonymous Identification schemes and the corresponding signer-ambiguous signature schemes can efficiently support identity escrow capabilities. As a result, we get an ID Escrow and a group signature scheme with the following nice features. (For concreteness, we concentrate on group signatures below.) First, just like in current state-of-the-art group signature schemes, the signature is of constant size. Second, a user can join any group by simply telling the group manager about its public key: no expensive interactive protocols, where the user will "get a special certificate" have to be run. Thus, the group manager only needs to decide if the user can join the group, and periodically certify the "current" public key of the group. In other words, we can imagine a simple bulletin board, where the group manager periodically publishes the (certified) group public key, the description of the group, and potentially the history of how the public key evolved (which is very handy for incremental *Ad Hoc* Anonymous Identification schemes; *cf.* Section 4.3.5). From this information, each member of the group can figure out its group secret key and sign arbitrary many messages efficiently. (Of course,

when signing a message the signer should also include the certified version of the current group key, so that "old" signatures do not get invalidated when the group key changes.)

Bibliography

[1] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *Advances in Cryptology—Crypto'00*, volume 1880 of *Lecture Notes in Computer Science*, pages 255–270, Berlin, 2000. Springer.

[2] G. Ateniese and B. de Medeiros. Efficient group signatures without trapdoors. In *Advances in Cryptology—ASIACRYPT '03*, volume 2894 of *LNCS*, pages 246–268. Springer, 2002.

[3] G. Ateniese and G. Tsudik. Some open issues and new directions in group signatures. In *Financial Cryptography (FC '99)*, volume 1648 of *LNCS*, pages 196–211. Springer, 1999.

[4] N. Barić and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *Advances in Cryptology—EUROCRYPT '97*, volume 1233 of *LNCS*, pages 480–494. Springer, 1997.

[5] M. Bellare, M. Fischlin, S. Goldwasser, and S. Micali. Identification protocols secure against reset attacks. In *Advances in Cryptology—EUROCRYPT '01*, volume 2045 of *LNCS*, pages 495–511. Springer, 2001.

[6] M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In *Advances in Cryptology—EUROCRYPT '03*, volume 2656, pages 630–648. Springer, 2003.

[7] M. Bellare and G. Neven. Transitive signatures based on factoring and RSA. In *Advances in Cryptology—AsiaCrypt'02*, volume 2501 of *Lecture Notes in Computer Science*, pages 397–414, Berlin, 2002. Springer-Verlag. Full version available at `http://eprint.iacr.org/2004/215/`.

[8] M. Bellare and A. Palacio. GQ and Schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks. In *Advances in Cryptology—Crypto'02*, volume 2442 of *Lecture Notes in Computer Science*, pages 162–177, Berlin, 2002. Springer-Verlag.

[9] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *First ACM Conference on Computer and Communications Security*, pages 62–73, Fairfax, VA, 1993.

[10] M. Bellare and P. Rogaway. The exact security of digital signatures—how to sign with RSA and Rabin. In *Advances in Cryptology—Eurocrypt 1996*, volume 1070 of *Lecture Notes in Computer Science*, pages 399–416, Berlin, 1996. Springer-Verlag.

[11] M. Bellare and R. Sandhu. The security of practical two-party RSA signature schemes. Technical Report 2001/060, IACR Cryptology ePrint Archive, 2001. Available at `http://eprint.iacr.org/2001/060/`.

[12] J. Benaloh and M. de Mare. One-way accumulators: a decentralized alternative to digital signatures. In *Advances in Cryptology—EUROCRYPT '93*, volume 765 of *LNCS*, pages 274–285. Springer, 1993.

[13] D. Bleichenbacher. Generating ElGamal signatures without knowing the secret key. In *Advances in Cryptology—EuroCrypt'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 10–18, Berlin, 1996. Springer-Verlag.

[14] M. Blum. Coin flipping by telephone. In *IEEE Spring COMPCOM*, pages 133–137, 1982.

[15] A. Boldyreva. Efficient threshold signature, multisignature and blind signature schemes based on the gap-diffie-hellman-group signature scheme. In *Public Key Cryptography—PKC'03*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46, Berlin, 2003. Springer-Verlag. Full version available at `http://eprint.iacr.org/2002/118/`.

[16] D. Boneh and M. Franklin. Anonymous authentication with subset queries. In *ACM Conference on Computer and Communications Security—CCS '99*, pages 113–119. ACM Press, 1999.

[17] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In *CRYPTO*, Aug. 2001.

[18] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *Advances in Cryptology—EUROCRYPT '03*, volume 2656 of *LNCS*, pages 416–432. Springer, 2003.

[19] F. Boudot. Efficient proofs that a commited number lies in an interval. In *Advances in Cryptology—EUROCRYPT '00*, volume 1807 of *LNCS*, pages 431–444. Springer, 2000.

[20] C. Boyd. Digital multisignatures. In *IMA Conference on Cryptography and Coding*, pages 241–246. Oxford University Press, 1989.

[21] E. Bresson and J. Stern. Efficient revocation in group signatures. In *Public Key Cryptography (PKC '01)*, volume 1992 of *LNCS*, pages 190–206. Springer, 2001.

[22] E. Bresson, J. Stern, and M. Szydlo. Threshold ring signatures and applications to ad-hoc groups. In *CRYPTO 2002*, volume 2442 of *LNCS*, pages 465–480. Springer, 2002.

[23] J. Camenisch. Efficient and generalized group signatures. In *Advances in Cryptology—EUROCRYPT '97*, volume 1233 of *LNCS*, pages 465–479. Springer, 1997.

[24] J. Camenisch and A. Lysyanskaya. An identity escrow scheme with appointed verifiers. In *Advances in Cryptology—CRYPTO '01*, volume 2139 of *LNCS*, pages 388–407. Springer, 2001.

[25] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and applications to efficient revocation of anonymous credentials. In *Advances in Cryptology—CRYPTO '02*, volume 2442 of *LNCS*, pages 61–76. Springer, 2002.

[26] J. Camenisch and M. Michels. A group signature scheme with improved efficiency. In *Advances in Cryptology—ASIACRYPT '98*, volume 1514 of *LNCS*, pages 160–174. Springer, 1998.

[27] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. Full length version of extended abstract in CRYPTO'03, available at: `http://shoup.net/papers/`, 2003.

[28] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In *Advances in Cryptology—CRYPTO '97*, volume 1294 of *LNCS*, pages 410–424. Springer, 1997.

[29] D. Chaum and E. van Heyst. Group signatures. In *Advances in Cryptology—EUROCRYPT '91*, volume 547 of *LNCS*, pages 257–265. Springer, 1991.

[30] L. Chen and T. P. Pedersen. New group signature schemes (extended abstract). In *Advances in Cryptology—EUROCRYPT 94*, volume 950 of *LNCS*, pages 171–181. Springer, 1994.

[31] J.-S. Coron. Optimal security proofs for PSS and other signature schmes. In *Advances in Cryptology—EuroCrypt'02*, volume 2332 of *Lecture Notes in Computer Science*, pages 272–287, Berlin, 2002. Springer-Verlag.

[32] R. Cramer. *Modular Design of Secure yet Practical Cryptographic Protocols*. PhD thesis, CWI and University of Amsterdam, November 1996.

[33] R. Cramer, I. B. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in*

Cryptology—Crypto'94, volume 839 of *Lecture Notes in Computer Science*, pages 174–187, Berlin, 1994. Springer.

[34] I. Damgård and J. Nielsen. Perfect hiding and perfect binding commitment schemes with constant expansion factor. In *Advances in Cryptology—Crypto'02*, volume 2442 of *Lecture Notes in Computer Science*, pages 581–596, Berlin, 2002. Springer-Verlag.

[35] A. De Santis, G. Di Crescenzo, and G. Persiano. Communication-efficient anonymous group identification. In *5th ACM Conference on Computer and Communications Security*, pages 73–82. ACM Press, 1998.

[36] A. De Santis, G. Di Crescenzo, G. Persiano, and M. Yung. On monotone formula closure of SZK. In *35th Annual Symposium on Foundations of Computer Science*, pages 454–465. IEEE Computer Society Press, 1994.

[37] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *Advances in Cryptology—Crypto'89*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315, Berlin, 1989. Springer-Verlag.

[38] Y. Dodis, J. Katz, S. Xu, and M. Yung. Strong key-insulated signature schemes. In *Public Key Cryptography—PKC'03*, volume 2567 of *Lecture Notes in Computer Science*, pages 130–144, Berlin, 2003. Springer-Verlag.

[39] Y. Dodis, A. Kiayias, A. Nicolosi, and V. Shoup. Anonymous identification in *ad hoc* groups. In *Advances in Cryptology—EuroCrypt'04*, volume 3027 of *LNCS*, pages 609–626, Berlin, 2004. Springer-Verlag.

[40] U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, 1988.

[41] U. Feige and A. Shamir. Zero knowledge proofs of knowledge in two rounds. In *Advances in Cryptology—Crypto'89*, volume 435 of *Lecture Notes in Computer Science*, pages 526–544, Berlin, 1989. Springer.

[42] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology—CRYPTO '86*, volume 263 of *LNCS*, pages 186–194. Springer, 1986.

[43] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology—Crypto'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Berlin, 1987. Springer-Verlag.

[44] FIPS 180-1. *Secure Hash Standard*. U.S. Department of Commerce/N.I.S.T., Springfield, VA, April 1995.

[45] FIPS 186. *Digital Signature Standard*. U.S. Department of Commerce/N.I.S.T., Springfield, VA, May 1994.

[46] Y. Frankel, P. Gemmell, P. MacKenzie, and M. Yung. Proactive RSA. In *Advances in Cryptology—Crypto'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 440–454, Berlin, 1997. Springer-Verlag.

[47] M. Freedman and A. Nicolosi. Efficient private techniques for verifying social proximity. In *Sixth International Workshop on Peer-to-Peer Systems (IPTPS '07)*, 2007.

[48] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *EUROCRYPT*, May 2004.

[49] R. Ganesan. Yaksha: Augmenting kerberos with public-key cryptography. In *Proceedings of the ISOC Network and Distributed Systems Security Symposium*, pages 132–143, 1995.

[50] S. Garriss, M. Kaminsky, M. J. Freedman, B. Karp, D. Mazières, and H. Yu. RE: Reliable email. In *Proc. NSDI*, May 2006.

[51] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 33(4), Oct. 1986.

[52] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proc. STOC*, May 1987.

[53] O. Goldreich, S. Micali, and A. Wigderson. Proof that yield nothing bur their validity or all languages in np have zero-knowledge proof systems. *Journal of the ACM*, 38(3):691–729, 1991.

[54] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[55] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal of Computing*, 17(2):281–308, 1988.

[56] L. Guillou and J.-J. Quisquater. A "paradoxical" identity-based signature scheme resulting from zero-knowledge. In *Advances in Cryptology—Crypto'88*, volume 403 of *Lecture Notes in Computer Science*, pages 216–231, Berlin, 1988. Springer-Verlag.

[57] A. Herzberg, M. Jacobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive public key and signature schemes. In *Fourth ACM Conference on Computer and Communication Security*, pages 100–110. ACM, 1997.

[58] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, 1988.

[59] G. Itkis and L. Reyzin. Forward-secure signature with optimal signing and verifying. In *Advances in Cryptology—Crypto'01*, volume 2139 of *Lecture Notes in Computer Science*, pages 332–354, Berlin, 2001. Springer-Verlag.

[60] G. Itkis and L. Reyzin. SiBIR: Signer-base intrusion-resilient signatures. In *Advances in Cryptology—Crypto'02*, volume 2442 of *Lecture Notes in Computer Science*, pages 499–514, Berlin, 2002. Springer-Verlag.

[61] A. Joux. A one round protocol for tripartite Diffie-Hellman. In *Proc. ANTS*, July 2000.

[62] A. Kiayias, Y. Tsiounis, and M. Yung. Traceable signatures. In *Advances in Cryptology—EUROCRYPT 04*, LNCS. Springer, 2004.

[63] J. Kilian and E. Petrank. Identity escrow. In *Advances in Cryptology—CRYPTO '98*, volume 1462 of *LNCS*, pages 169–185. Springer, 1998.

[64] J. Kong, P. O. Boykin, B. Rezaei, N. Sarshar, and V. Roychowdhury. Let your cyberalter ego share information and manage spam, May 2005. `http://arxiv.org/abs/physics/0504026`.

[65] P. MacKenzie and M. Reiter. Delegation of cryptographic servers for capture-resilient devices. In *Eight ACM Conference on Computer and Communication Security*, pages 10–19. ACM, 2001. Full version available from `ftp://dimacs.rutgers.edu/pub/dimacs/TechnicalReports/` a DIMACS Technical Report 2001-37.

[66] P. MacKenzie and M. Reiter. Networked cryptographic devices resilient to capture. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, 2001.

[67] P. MacKenzie and M. Reiter. Two-party generation of DSA signature. In *Advances in Cryptology—Crypto'01*, volume 2139 of *Lecture Notes in Computer Science*, pages 137–154, Berlin, 2001. Springer-Verlag.

[68] D. Mazières, M. Kaminsky, M. F. Kaashoek, and E. Witchel. Separating key management from file system security. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, pages 124–139, Kiawa Island, SC, 1999. ACM.

[69] M. Naor. Deniable ring authentication. In *Advances in Cryptology—CRYPTO '02*, volume 2442 of *LNCS*, pages 481–498. Springer, 2002.

[70] A. Nicolosi, M. Krohn, Y. Dodis, and D. Mazières. Proactive two-party signatures for user authentication. In *Proceedings of the Tenth Network and Distributed System Security Symposium (NDSS '03)*, pages 233–248. Internet Society (ISOC), 2003.

[71] R. Ostrovsky and M. Yung. How to withstand mobile virus attacks. In *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing*, pages 51–59, 1991.

[72] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000.

[73] R. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In *Advances in Cryptology—ASIACRYPT '01*, volume 2248 of *LNCS*, pages 552–565. Springer, 2001.

[74] C. Schnorr. Efficient identification and signature for smart cards. In *Advances in Cryptology—Crypto'89*, volume 435 of *Lecture Notes in Computer Science*, pages 235–251, Berlin, 1990. Springer-Verlag.

[75] C. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[76] J. G. Steiner, B. C. Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. In *Proceedings of the Winter 1988 USENIX*, pages 191–202, Dallas, TX, 1988. USENIX.

[77] G. Tsudik and S. Xu. Accumulating composites and improved group signing. In *Advances in Cryptology—ASIACRYPT '03*, volume 2894 of *LNCS*, pages 269–286. Springer, 2003.

[78] H. C. Williams. A modification of the RSA public-key encryption procedure. *IEEE Transactions on Information Theory*, IT-26(6):726–729, 1980.

[79] T. Wu. The secure remote password protocol. In *Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium*, pages 97–111, San Diego, CA, March 1998.

[80] A. Yao. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167, 1986.

[81] T. Ylönen. SSH: Secure login connections over the Internet. In *Proceedings of the 6th USENIX Security Symposium*, pages 37–42, San Jose, CA, 1996.