An extended abstract of this paper appears in *Advances in Cryptology – Crypto 99 Proceedings*, Lec. Notes in Comp. Sci. Vol. ??, M. Wiener ed., Springer-Verlag, 1999. This is the full version.

A Forward-Secure Digital Signature Scheme

Mihir Bellare*   Sara K. Miner†

July 13, 1999

Abstract

We describe a digital signature scheme in which the public key is fixed but the secret signing key is updated at regular intervals so as to provide a *forward security* property: compromise of the current secret key does not enable an adversary to forge signatures pertaining to the past. This can be useful to mitigate the damage caused by key exposure without requiring distribution of keys. Our construction uses ideas from the Fiat-Shamir and Ong-Schnorr identification and signature schemes, and is proven to be forward secure based on the hardness of factoring, in the random oracle model. The construction is also quite efficient.

Keywords: Digital signatures, identification schemes, forward security.

* Department of Computer Science and Engineering, Mail Code 0114, University of California at San Diego, 9500 Gilman Drive, La Jolla, CA 92093, USA. E-mail: `mihir@cs.ucsd.edu` ; Web page: `http://www-cse.ucsd.edu/users/mihir`. Supported in part by a 1996 Packard Foundation Fellowship in Science and Engineering, and NSF CAREER Award CCR-9624439.

† Department of Computer Science and Engineering, Mail Code 0114, University of California at San Diego, 9500 Gilman Drive, La Jolla, CA 92093, USA. E-mail: `sminer@cs.ucsd.edu` ; Web page: `http://www-cse.ucsd.edu/users/sminer`. Supported in part by a National Physical Sciences Consortium Fellowship and the above-mentioned grants of Bellare.

Contents

1 Introduction

This paper presents a digital signature scheme having a novel security property, namely "forward security." This is a means to mitigate the damage caused by key exposure.

1.1 The key exposure problem

In practice the greatest threat against the security of a digital signature scheme is exposure of the secret (signing) key, due to compromise of the security of the underlying system or machine storing the key. The danger of successful cryptanalysis of the signature scheme itself is hardly as great as the danger of key exposure, as long as we stick to well-known schemes and use large security parameters.

The most widely considered solution to the problem of key exposure is distribution of the key across multiple servers via secret sharing [20, 5]. There are numerous instantiations of this idea including threshold signatures [7] and proactive signatures [15]. Distribution however is quite costly. While a large corporation or a certification authority might be able to distribute their keys, the average user, with just one machine, does not have this option. Thus while we expect digital signatures to be very widely used, we do not expect most people to have the luxury of splitting their keys across several machines. Furthermore even when possible, distribution may not provide as much security as one might imagine. For example, distribution is susceptible to "common-mode failures:" a system "hole" that permits break-in might be present on all the machines since they are probably running a common operating system, and once found, all the machines can be compromised.

Other ways of protecting against key exposure include use of protected hardware or smartcards, but these can be costly or impractical.

1.2 Forward secure signatures

The goal of forward security is to protect some aspects of signature security against the risk of exposure of the secret signing key, but in a simple way, in particular without requiring distribution or protected storage devices, and without increasing key management costs.

How is it possible to preserve any security in the face of key exposure without distribution or protected devices? Obviously, we cannot hope for total security. Once a signing key is exposed, the attacker can forge signatures. The idea of "forward security" is however that a distinction can be made between the security of documents pertaining to (meaning dated in) the past (the time prior to key exposure) and those pertaining to the period after key exposure.

THE KEY EVOLUTION PARADIGM. A user begins, as usual, by registering a public key PK and keeping private the corresponding secret key, which we denote SK_0. The time during which the public key PK is desired to be valid is divided into periods, say T of them, numbered $1, \ldots, T$. While the public key stays fixed, the user "evolves" the secret key with time. Thus in each period, the user produces signatures using a different signing key: SK_1 in period 1, SK_2 in period 2, and so on. The secret key in period i is derived as a function of the one in the previous period; specifically, when period i begins, the user applies a public one-way function h to SK_{i-1} to get SK_i. At that point, the user also deletes SK_{i-1}. Now an attacker breaking in during period i will certainly get key SK_i, but not the previous keys $SK_0, \ldots, SK_{i-1}$, since they have been deleted. (Furthermore the attacker cannot obtain the old keys from SK_i because the latter was a one-way function of the previous key.) The key evolution paradigm is illustrated in Figure 1.

$$\begin{array}{ccc|c|c|c|c|}
 & & & \text{Period 1} & \text{Period 2} & \cdots \quad \cdots & \text{Period } T \\
SK & \xrightarrow{h} & & SK_1 \quad \xrightarrow{h} & SK_2 \quad \xrightarrow{h} & \cdots \quad \cdots \xrightarrow{h} & SK_T
\end{array}$$

Figure 1: Paradigm for a forward secure signature scheme: Secret key in a given period is a one-way function h of the secret key in the previous period.

A signature always includes the value j of the time period during which it was produced, so that it is viewed as a pair $\langle j, \zeta \rangle$. The verification algorithm takes the (fixed) public key PK, a message and candidate signature, and verifies that the signature is valid in the sense that it was produced by the legitimate user *in the period indicated in the signature.* We stress that although the user's secret key evolves with time, the public key stays fixed throughout, so that the signature verification process is unchanged, as are the public key certification and management processes.

The number of periods and the length of each period are parameters of choice. For example, we might want to use the scheme under a certain public key for one year, with daily updates, in which case $T = 365$ and each period has length one day.

SECURITY BENEFITS. The security benefit of this key evolution paradigm is that loss of the current secret key will not enable the adversary to forge signatures with a "date" prior to the one at which key exposure occurred. This "date" has a precise meaning: it is the value of the period during which the signature is (claimed to be) produced, which as indicated above is always included in the signature itself. This can be viewed as a means of protecting the authenticity of past transactions even in the event of key exposure. It is quite a useful property and can mitigate the damage of key exposure in several ways.

The following example illustrates. Suppose Alice is a notary public having public key PK, and using a normal (ie. not forward-secure) signature scheme. On January 1st, 1999, Bob brings to her a document D which she notarizes, by signing D to produce a signature σ. (We assume that the date, January 1st, 1999, is part of D.) Bob expects to be able to use the document D for a long time to come. Meaning even a few years later, he would like to be able to take D to a verifier, say a court of law, and have the signature accepted. Now suppose Alice's secret signing key is compromised a little after she signed D, say on February 1st, 1999. She discovers the fact and revokes her public key. Now, the notarized document D will no longer be accepted. Indeed, it would be unwise to accept it, because if Alice's secret key is no longer secure, Bob could have produced a signature on D himself. The fact that D is dated "in the past" makes no difference: it is just a text that Bob could sign if he knew Alice's secret key. This reduces the quality of the service that Alice, as a notary, can provide.

Now suppose instead that Alice uses a forward-secure signature scheme with key updates taking place daily. The signature of D produced on January 1st, 1999, has the form $\langle 1, \zeta \rangle$ where ζ is a tag for document D and "1" represents January 1st, 1999. If Alice's security is compromized on February 1st, she looses SK_{32}. She revokes her public key only for any period following February 1st, meaning all signatures of the form $\langle j, \zeta \rangle$ where $j < 32$ are still to be accepted. Now, Bob can still use Alice's signature of document D as a guarantee, and it will still be accepted. Rightfully so, because possession of SK_{32} does not enable forgery of tags of the form $\langle j, \zeta \rangle$ where $j < 32$. This leads to a higher quality notary service.

RELATION TO TIME-STAMPING. Time-stamping signed documents via a trusted time stamping authority (cf. [14]) can also provide a similar kind of security, but this requires that one make use of such an authority, which is costly in various ways. Forward security may be viewed as providing a certain kind of time-stamp, namely one that is secure against forgery by an adversary who obtains the current secret key. (However it assumes the signer is honest since the signer could of course "forge" time-stamps if it wanted by simply not deleting previous keys.)

History. The term (perfect) "forward secrecy" was first used in [13] in the context of session key exchange protocols, and later in [8]. The basic idea, as described in [8], is that compromise of long-term keys does not compromise past session keys, meaning that past actions are protected in some way against loss of the current key, the same basic idea as here in a different context. The above paradigm and the idea of a digital scheme with forward security were suggested by Ross Andersen in an invited lecture at the ACM CCS conference [1]. He left as an open problem to find such a scheme. In this paper we provide the first solution. We also provide a formal adversarial model and notion of security with respect to which the security of schemes can be assessed. In particular we suggest the inclusion of the current date in the signature and the use of the claimed date as the determinant of whether something is past or present from the forward security point of view.

1.3 Construction of a forward secure digital signature scheme

First ideas. It is trivial to design a forward secure signature scheme if you allow any of scheme parameters to grow proportionally with the number T of time periods over which the scheme is supposed to be valid; specifically, if the size of the public key, secret key, or the signature itself is allowed to be proportional to T. We briefly describe these methods in Section 3. However, any such method is impractical.

The first reasonable solution, also described in Section 3, involves the use of binary tree based certification chains, and results in secret keys and signatures of size linear in $\lg(T)$. Ideally however we would like a solution in which the size of the public key, secret key, and signature does not depend on the number of time periods. But achieving forward security in this setting does not seem so easy. A natural starting idea is to try to find a suitable key evolution mechanism for standard RSA or El Gamal type schemes, but we were unable to find a secure one in either case.

Our scheme. To get a forward-secure digital signature scheme of the desired kind, we turn to a different paradigm: construction of signature schemes based on identification schemes. We present a forward secure digital signature scheme based on ideas underlying the Fiat-Shamir [10] identification scheme (in turn based on the zero-knowledge proof of quadratic residuosity of Goldwasser, Micali and Rackoff [11]) and the Ong-Schnorr [17] identification scheme. The feature of these schemes that is crucial to enable secure key evolution is that even the signer does not need to know the factorization of the modulus on which the scheme is based, but just the square roots of some public values. We evolve the secret key via squaring, which is a one-way function in this setting.

Notion of security. To provide assurance that our scheme has the forward security property, we first provide a formal definition of forward security for digital signatures, extending the notion of security for standard digital signatures from [12] to allow for key exposure attacks. Our model allows the adversary to mount a chosen-message attack, then expose the signing key at some current period j of its choice. It is successful if it can forge a signature of the form $\langle i, \zeta \rangle$ for some message M where $i < j$ pertains to a period prior to that of the key exposure. The scheme is secure if this task is computationally infeasible.

Forward security of our scheme. We then show that our scheme meets this notion of forward security assuming it is hard to factor Blum-Williams integers. This proof is in the random oracle model [3], meaning it assumes that a certain hash function used in the scheme has random behavior.

Our security analysis proceeds in two steps. We first define a notion of a "forward secure identification (ID) scheme," design such a scheme, and prove it secure based on the hardness of factoring Blum-Williams integers. Our signature scheme is obtained from the ID scheme by the usual transformation of ID schemes into signature schemes (in which the challenge is specified as a hash of the message and the commitment). We then show that this transformation preserves forward security. (The transformation is known to preserve security in the standard sense [18, 16], but here we are considering a new security feature.)

We stress one issue with regard to forward secure ID schemes: they are artificial constructs, in the sense that the security notion we put forth there, although mathematically viable, does not correspond to any real attack in a practical setting. (This is because identification is an on-line activity, unlike signature verification; one cannot return to the past and try to identify oneself.) But this does not matter in our setting, because for us, the ID scheme is simply a convenient abstraction that enables our construction of a forward secure signature scheme, and the latter is a real and useful object.

Our goal here has been to present the simplest possible scheme that has the forward security property, is reasonably efficient, and can be proven secure. Improvements and alternatives seem possible. In particular one could try to build a scheme only in the Ong-Schnorr style (rather than a combination of that with Fiat-Shamir as we do). This will reduce key sizes, but increase computation time, and the analysis (one could try to extend [21]) seems more involved.

A NOTE ON SYNCHRONIZATION. One might imagine that a scheme using time periods in the way we do will impose a requirement for clock synchronization, arising from disagreements near the time period boundaries over what is truly the current time. However, in our signature scheme, discrepancies over time periods do not cause problems. This is due to the fact that the time period in which the message was signed is stamped on the message, and the verifier looks at this stamp to determine which verification process to use. That is, it doesn't matter what the time period of verification is; the signature is verified using the information from the time period when the signature actually *occurred.*

2 Definitions and basic facts

Here we provide definitions of what it means for a signature scheme to possess the forward security property. We first discuss the form of algorithms to specify such schemes, and then discuss security.

THE ALGORITHMS. We are interested in "key evolving" digital signature schemes, namely schemes that operate over a certain length of time that is divided into periods, with the secret key evolving across periods. In addition to the usual processes that specify a signature scheme (namely the key generation, signing and verifying processes) we need a key update process.

Definition 2.1 [Key-evolving signature scheme] A *key-evolving digital signature scheme* $\mathsf{KE\text{-}SIG} = (KG, Upd, Sgn, Vf)$ consists of four algorithms:

(1) The *key generation* algorithm KG takes as input a security parameter $k \in \mathsf{N}$, the total number of periods T over which the scheme will operate, and possibly other parameters, to return a *base public key* PK and corresponding *base secret key* SK_0. The algorithm is probabilistic.

(2) The *secret key update* algorithm Upd takes as input the secret signing key SK_{j-1} of the previous period to return the secret signing key SK_j of the current period. The algorithm is usually deterministic.

(3) The *signing* algorithm *Sgn* takes the secret signing key SK_j of the current period and a message M to return a signature of M for period j. We write $\langle j, \zeta \rangle \stackrel{R}{\leftarrow} Sgn_{SK_j}(M)$. The algorithm might be probabilistic. The signature is always a pair consisting of the value j of the current period and a tag ζ.

(4) The *verification* algorithm *Vf* takes the public key PK, message M and candidate signature $\langle j, \zeta \rangle$ to return a bit, with 1 meaning accept and 0 meaning reject. We write $b \leftarrow Vf_{PK}(M, \langle j, \zeta \rangle)$. The algorithm is typically deterministic.

We say that $\langle j, \zeta \rangle$ is a *valid* signature of M for period j if $Vf_{PK}(M, \langle j, \zeta \rangle) = 1$. It is required that a signature of M generated via $Sgn_{SK_j}(M)$ be a valid signature of M for period j. It is assumed that the secret key SK_j for period $j \in \{1, \ldots, T\}$ always contains the value j itself and also always contains the value T of the total number of periods. Finally we adopt the convention that SK_{T+1} is the empty string and that Upd_{SK_T} returns SK_{T+1}. ▮

Sometimes we work in a random oracle model. In this case the algorithms above additionally have oracle access to a public hash function H which in the security analyses is assumed random.

OPERATION. A user begins by generating a key pair (PK, SK_0) via the key generation algorithm. The public key is treated identically to that in a standard signature scheme in terms of registration, key management, and certification. The user stores SK_0. At the start of the first time period, the user runs the secret key update algorithm to update his key: $SK_1 \leftarrow Upd(SK_0)$. Having done this, it deletes SK_0 from memory. For the continuation of time period one it signs messages using SK_1. This means that to generate a signature of a message M during period 1, it runs $Sgn_{SK_1}(M)$ to get a signature $\langle 1, \zeta \rangle$. When the second period begins, it sets $SK_2 \leftarrow Upd(SK_1)$, deletes SK_1, and uses SK_2 to sign. And so on. In any time period j, only SK_j is available to an adversary who might break in. Notice that verification is always performed with respect to the fixed public key PK; that latter does not change. However, the verification process is "aware" of the time period during which a signature is claimed to have been generated because this value is included in the signature itself.

SECURITY. We wish to assess the forward security of a key-evolving signature scheme. To do this effectively we must first pin down an appropriate model; what can the adversary do, and when is it declared successful? We extend the notion of Goldwasser, Micali and Rivest [12] (security against existential forgery under adaptive chosen-message attack) to take into account the obtaining of a key via break-in.

The adversary knows the total number of time periods, the current time period, and the user's public key PK. Recall the goal is that even under exposure of the current secret key it should be computationally infeasible for an adversary to forge a signature "with respect to" a previous secret key. The possibility of key exposure is modeled by allowing the adversary a "break-in" move, during which it can obtain the secret key SK_j of the current period j. The adversary is then considered successful if it can create a valid forgery where the signature has the form $\langle i, (Y, Z) \rangle$ for some $i < j$, meaning is dated for a previous time period. The model is further augmented to allow the adversary a chosen-message attack prior to its break-in. In that phase, the adversary gets to obtain genuine signatures of messages of its choice, under the keys $SK_1, SK_2, \ldots$ in order, modeling the creation of genuine signatures under the key evolution process. The adversary stops the chosen-message attack phase at a point of its choice and then gets to break-in.

For the formalization, fix a key evolving signature scheme $\mathsf{KE\text{-}SIG} = (KG, Upd, Sgn, Vf)$. The adversary F is viewed as functioning in three stages: the chosen message attack phase (cma); the break-in phase ($\mathsf{breakin}$); and the forgery phase (forge). Its success probability in breaking the forward-security of the signature scheme is evaluated by the following experiment. We write $k, \ldots, T$ as the arguments to the key generation algorithm to indicate the possible presence of extra arguments. (For example in our scheme of Figure 2, the process depends on a parameter l.)

Experiment F-Forge($\mathsf{KE\text{-}SIG}, F$)
$(PK, SK_0) \stackrel{R}{\leftarrow} KG(k, \ldots, T)$
$i \leftarrow 0$
`Repeat`
$j \leftarrow j+1$; $SK_j \leftarrow Upd(SK_{j-1})$; $d \leftarrow F^{Sgn_{SK_j}(\cdot)}(\mathsf{cma}, PK)$
`Until` $(d = \mathsf{breakin})$ or $(j = T)$
`If` $d \neq \mathsf{breakin}$ and $j = T$ `then` $j \leftarrow T+1$
$(M, \langle b, \zeta \rangle) \leftarrow F(\mathsf{forge}, SK_j)$
`If` $Vf_{PK}(M, \langle b, \zeta \rangle) = 1$ and $1 \le b < j$ and M was not queried of $Sgn_{SK_b}(\cdot)$ in period b
`then return` 1 `else return` 0

It is understood above that in running F we first pick and fix coins for it, and also that we preserve its state across its various invocations. The chosen-message attack reflects the way the signature scheme is used. The adversary first gets access to an oracle for generating signatures under SK_1. It queries this as often as it wants, and indicates it is done by outputting some value d. As long as d is not the special value breakin, we move into the next stage, providing the adversary an oracle for signing under the next key. Note that the process is strictly ordered; once an adversary gives up the oracle for signing under SK_j, by moving into the next phase or breaking-in, it cannot obtain access to that oracle again. At some point the adversary decides to use its break-in privelege, and is returned the key SK_j of the stage in which it did this. (If it does not break-in by the last period, we give it the key SK_j where $j = T+1$. Recall that by definition this key is just the empty string.) It will now try to forge signatures under SK_b for some $b < j$ and is declared successful if the signature is valid and the message is new.

Following the concrete security paradigm used in [4], we associate to the scheme an *insecurity function* whose value is the maximum probability of being able to break the scheme, the maximum being over all adversary strategies restricted to resource bounds specified as arguments to the insecurity function.

Definition 2.2 [Forward-security] Let $\mathsf{KE\text{-}SIG} = (KG, Upd, Sgn, Vf)$ be a key-evolving signature scheme, and F an adversary attacking its forward security as described above. We let $\mathbf{Succ}^{\text{fwsig}}(\mathsf{KE\text{-}SIG}[k, \ldots, T], F)$ denote the probability that experiment F-Forge($\mathsf{KE\text{-}SIG}[k, \ldots, T], F$) returns 1. (This is the probability that the adversary F is successful in breaking FSIG in the forward security sense.) Then the insecurity function of KE-SIG is defined as

$$\mathbf{InSec}^{\text{fwsig}}(\mathsf{KE\text{-}SIG}[k, \ldots, T]; t, q_{\text{sig}}) = \max_F \{ \mathbf{Succ}^{\text{fwsig}}(\mathsf{KE\text{-}SIG}[k, \ldots, T], F) \} .$$

The maximum here is over all F for which the following are true: the execution time of the above experiment, plus the size of the code of F, is at most t; F makes a total of at most q_{sig} queries to the signing oracles across all the stages. ∎

Note the execution time is that of the entire experiment F-Forge($\mathsf{KE\text{-}SIG}[k, \ldots, T], F$), not just that of F. So it includes in particular the time to compute answers to oracle queries.

The smaller the value taken by the insecurity function, the more secure the scheme. Our goal will be to upper bound the values taken by this insecurity function. Under our concrete security approach, there is no formal notion of a scheme being "secure"; every scheme simply has some associated insecurity. Informally we say a scheme is secure if the insecurity function is "low" for reasonably "high" values of the resource parameters it gets as arguments. We recover the usual "asymptotic" notions by saying a scheme is secure if the success probability of any probabilistic, polynomial time adversary is negligible.

Actually our main scheme is in the random oracle model, and we will have to augment the definition to count also the number of queries q_{hash} made to the random oracle in the experiment.

Remark 2.3 [Forward-security implies normal security] It is easy to see that a scheme that is forward-secure also meets the standard notion of security for digital signatures of [12]. The model above includes adversaries that do not exploit their break-in privelege; these correspond to normal adversaries. Such an adversary never outputs $d = \mathsf{breakin}$. By default it receives the empty string SK_{T+1} as the key, meaning gets nothing, and then wins if it can forge successfully relative to any time period $b \leq T$. Exploiting the break-in privelege however leads to new attacks, and that is what we want to address.

3 Some simple solutions

We summarize several simple ways to design a forward secure signature scheme. The first three methods result in impractical schemes: some parameter (the public key size, the secret key size, or the signature size) grows linearly with the number T of time periods over which the public key is supposed to be valid. The methods are nonetheless worth a brief look in order to better understand the problem and to lead into the fourth method. The latter, which we call the tree scheme, is a reasonable binary certification tree based solution in which key and signature sizes are logarithmic in T. Nonetheless it would be preferable if even this dependency is avoided, which is accomplished by our main scheme presented in Section 4.

In the following we make use of some fixed standard digital signature scheme whose key generation, signing and verifying algorithms are denoted KG, SGN and VF respectively. The descriptions that follow are informal. However the schemes can be described formally in terms of the definitions in Section 2, and can also be proven secure. We do not elaborate because these schemes are not really practical, and prefer to move quickly to our main scheme of Section 4.

3.1 Long public and secret keys

The signer generates T pairs $(p_1, s_1), \ldots, (p_T, s_T)$ of matching public and secret keys for the standard scheme, via T executions of KG. He sets the public key of the key evolving scheme to $PK = (p_1, \ldots, p_T)$ and his starting (base) secret key for the key evolving scheme to $SK_0 = (s_0, s_1, \ldots, s_T)$ where s_0 is the empty string. Upon entering period j the signer deletes s_{j-1}, so that he is left with key $SK_j = (s_j, \ldots, s_T)$. The signature of a message m in period j is $\langle j, \mathrm{SGN}_{s_j}(m)\rangle$. A signature $\langle j, \zeta\rangle$ on a message m is verified by checking that $\mathrm{VF}_{p_j}(m, \zeta) = 1$. This method clearly provides forward security, but the size of the keys (both public and secret) of the key evolving scheme depends on T, which is not desirable.

3.2 Long secret key only

Andersen [1] suggested the following variant which results in a short public key but still has a secret key of size proportional to T. Generate T key-pairs as above, and an additional pair (p, s). Let $\sigma_j = \mathrm{SGN}_s(j \| p_j)$ be a signature (with respect to p) of the value j together with the j-th public key, for $j = 1, \ldots, T$. This done, delete s. The public key of the key evolving scheme is p, and the signer's starting (base) secret key for the key evolving scheme is $(s_0, \sigma_0, s_1, \sigma_1, \ldots, s_T, \sigma_T)$ where s_0, σ_0 are set to the empty string. Upon entering period j the signer deletes s_{j-1}, σ_{j-1}, so that he is left with secret key $(s_j, \sigma_j, \ldots, s_T, \sigma_T)$. The signature of a message m in period j is $\langle j, (\mathrm{SGN}_{s_j}(m), p_j, \sigma_j)\rangle$. A signature $\langle j, (\alpha, q, \sigma)\rangle$ on a message m is verified by checking that $\mathrm{VF}_q(m, \alpha) = 1$ and $\mathrm{VF}_p(j \| q, \sigma) = 1$. This method, like the above, clearly provides forward security. (Notice that it is crucial for the forward security that s be deleted as soon as the signatures of the subkeys have been generated.) Furthermore the public key has size independent of T. But the size of the secret key still depends on T.

3.3 Long signatures

The size of both the public and the secret key can be kept small by using certification chains, but this results in large signatures. The signer begins by generating a key pair (p_0, s_0) of the standard scheme. He sets the public key of the key evolving scheme to p_0 and the starting (base) secret key of the key evolving scheme to s_0. At each period a new key is generated and certified with respect to the previous key, which is then deleted. The certificate chain is included in the signature. To illustrate, at the start of period 1 the signer creates a new key pair (p_1, s_1), sets $\sigma_1 = \mathrm{SGN}_{s_0}(1 \| p_1)$, and deletes s_0. The signature of a message m in period 1 is $\langle 1, (\mathrm{SGN}_{s_1}(m), p_1, \sigma_1)\rangle$. A signature $\langle 1, (\alpha, q_1, \tau_1)\rangle$ on a message m is verified by checking that $\mathrm{VF}_{q_1}(m, \alpha) = 1$ and $\mathrm{VF}_{p_0}(1 \| q_1, \tau_1) = 1$. This continues iteratively, so that at the start of period $j \geq 2$ the signer, in possession of $p_1, \sigma_1, \ldots, p_{j-1}, \sigma_{j-1}$ and the secret key s_{j-1} of the previous period, creates a new key pair (p_j, s_j), sets $\sigma_j = \mathrm{SGN}_{s_{j-1}}(j \| p_j)$, and deletes s_{j-1}. The signature of a message m in period j is $\langle j, (\mathrm{SGN}_{s_j}(m), p_1, \sigma_1, \ldots, p_j, \sigma_j)\rangle$. A signature $\langle j, (\alpha, q_1, \tau_1, \ldots, q_j, \tau_j)\rangle$ on a message m is verified by checking that $\mathrm{VF}_{q_j}(m, \alpha) = 1$ and $\mathrm{VF}_{q_{i-1}}(i \| q_i, \tau_i) = 1$ for $i = 2, \ldots, j$ and $\mathrm{VF}_{p_0}(1 \| q_1, \tau_1) = 1$. Again, forward security is clearly provided. Furthermore both the public and the secret key are of size independent of T. But the size of a signature grows linearly with T. Also note that although the size of the signer's secret key has shrunk, the signer does have to store the list of public keys and their tags, which means it must use storage linear in T, which is not desirable.

3.4 Binary certification tree scheme

Andersen's scheme above can be viewed as building a certification tree of depth 1 and arity T, while the "long signature" solution just presented builds a standard certification chain, which is a tree of depth T and arity 1. Signature size is linear in the depth, and key size is linear in the arity. It is natural to form a "hybrid" scheme so as to get the best possible tradeoff between these sizes. Namely we use a binary tree with T leaves. Each key certifies its two children keys. The leaf keys are the actual message signing keys, one for each period, and the root public key is the public key of the key evolving scheme. The tree grows down from the root in a specific manner, with nodes being created dynamically. A node certifies its two children, and as soon as the two children nodes are created and certified, the secret key corresponding to the parent node is deleted. Appropriately done this results in a forward secure scheme with secret key size and signature size linear in $\lg(T)$, and fixed public key size. Furthermore the total amount of information (whether secret or not) stored at any time by the signer, and the computation time for key updates, are linear in $\lg(T)$.

One has to be a little careful to build the tree in the right way and delete keys at the right times, so let us give a few more details. For simplicity assume $T = 2^\tau$ is a power of two. Imagine a binary tree with T leaves. The root is at level 0 and the leaves are at level τ. The root is labeled with the empty string ε, and if a node has label the binary string w then its left child is labeled $w0$ and its right child is labeled $w1$. Associate to a node with label w a key pair $(p[w], s[w])$. (The entire tree never exists at any time, but is useful to imagine.) The public key of the key evolving scheme is $p[\varepsilon]$. Let $\langle i, n\rangle$ denote the binary representation of the integer $i-1$ as a string of exactly n bits.

In period $j \geq 1$ the signer signs data under $s[\langle j, \tau\rangle]$, and attaches a certification chain based on the path from leaf $\langle j, \tau\rangle$ to the root of the tree. During period j, a certain subtree $\mathcal{T}$ of the full tree exists. It consists of all nodes on the path from leaf $\langle j, \tau\rangle$ to the root, plus the right sibling of any of these nodes that is a left child of its parent. All childless nodes w in the subtree that are right children of their parents have their secret key $s[w]$ still attached; for all other nodes, it has been deleted. Notice that if a node w has $s[w]$ still "alive," its descendent leaves have names $\langle i, \tau\rangle$ with $i > j$, so correspond to future periods. When the signer enters period $j+1$ (here $j < T$) it must update its subtree. This involves (possibly) creation of new nodes, and deletion of old ones in such a way that the property of the subtree described above is preserved. The signer moves up from $\langle j, \tau\rangle$ (deleting this leaf node and its keys) and stops at the first node whose left (rather than right) child is on the path from $\langle j, \tau\rangle$ to the root. Call this node w. We know that the secret key of its right child $w1$ is alive. If $w1$ is a leaf, the update is complete. Else, create its children by picking new key pairs for each child, and delete the secret key at w. Then move left, and continue this process until a leaf is reached.

Remark 3.1 [Forward-secure signatures with any one-way function] The scheme presented here answers the theoretical question of whether it is possible to design a non-trivial forward-secure signature signature scheme assuming only the existence of a one-way function. (By non-trivial we mean that the size of keys and signatures can be at most polylog(T).) Indeed, the tree scheme uses only standard digital signatures, and these exist given any one-way function [19].

4 Our forward-secure signature scheme

This is our main scheme. We first specify the scheme and its basic properties. Then we analyze its security.

4.1 Description of our scheme

KEYS AND KEY GENERATION. The signer's public key contains a modulus N and l points $U_1, \ldots, U_l$ in Z_N^*. The corresponding *base secret key* SK_0 contains points $S_1, \ldots, S_l$ in Z_N^*, where S_j is a 2^{T+1}-th root of U_j for $j = 1, \ldots, T$. The signer generates the keys by running the key generation process $KG(k, l, T)$ shown in Figure 2, which takes as input the security parameter k determining the size of N, the number l of points in the keys, and the number T of time periods over which the scheme is to operate.

As the code indicates, the keys contain some sundry information in addition to that mentioned above. Specifically the number T of time periods is thrown into the public key. This enables the verifier to know its value, which might vary with different signers. It is also thrown into the secret key for convenience, as is the modulus N. The third component of the base secret key, namely "0", is there simply to indicate that this is the base key, in light of the fact that the key will be evolving later. The modulus is a Blum-Williams integer, meaning the product of two distinct primes each congruent to 3 mod 4. We refer to U_i as the i-th *component* of the public key.

The public key PK is treated like that of any ordinary signature scheme as far as registration, certification and key generation are concerenced. The base secret key SK_0 is stored privately. The factors p, q of N are deleted once the key generation process is complete, so that they are not available to an attacker that might later break into the system on which the secret key is stored.

`Algorithm` $KG(k, l, T)$
  Pick random, distinct $k/2$ bit primes p, q, each congruent to 3 mod 4
  $N \leftarrow pq$
  `For` $i = 1, \ldots, l$ `do` $S_i \stackrel{R}{\leftarrow} Z_N^*$; $U_i \leftarrow S_i^{2^{(T+1)}} \bmod N$ `EndFor`
  $SK_0 \leftarrow (N, T, 0, S_{1,0}, \ldots, S_{l,0})$; $PK \leftarrow (N, T, U_1, \ldots, U_l)$
  `Return` (PK, SK_0)

`Algorithm` $Upd(SK_{j-1})$ where $SK_{j-1} = (N, T, j-1, S_{1,j-1}, \ldots, S_{l,j-1})$ and $1 \le j \le T+1$
  `If` $j = T+1$ `then` return the empty string
  `Else`
    `For` $i = 1, \ldots, l$ `do` $S_{i,j} \leftarrow S_{i,j-1}^2 \bmod N$ `EndFor`
    $SK_j \leftarrow (N, T, j, S_{1,j}, \ldots, S_{l,j})$; `Return` SK_j
  `End If`

`Algorithm` $Sgn_{SK_j}^H(M)$ where $SK_j = (N, T, j, S_{1,j}, \ldots S_{l,j})$
  $R \stackrel{R}{\leftarrow} Z_N^*$; $Y \leftarrow R^{2^{(T+1-j)}} \bmod N$; $c_1 \ldots c_l \leftarrow H(j, Y, M)$; $Z \leftarrow R \cdot \prod_{i=1}^{l} S_{i,j}^{c_i} \bmod N$
  `Return` $\langle j, (Y, Z)\rangle$

`Algorithm` $Vf_{PK}^H(M, \langle j, (Y, Z)\rangle$ where $PK = (N, T, U_1, ..., U_l)$
  $c_1 \ldots c_l \leftarrow H(j, Y, M)$
  `If` $Z^{2^{(T+1-j)}} = Y \cdot \prod_{i=1}^{l} U_i^{c_i} \bmod N$ `Then` return 1 `Else` return 0

Experiment F-Forge($\mathsf{FSIG}[k, l, T], F$)
  Select H: $\{0,1\}^* \to \{0,1\}^l$ at random
  $(PK, SK_0) \stackrel{R}{\leftarrow} KG(k, l, T)$
  $i \leftarrow 0$
  `Repeat`
    $j \leftarrow j+1$; $SK_j \leftarrow Upd(SK_{j-1})$; $d \leftarrow F^{H, Sgn_{SK_j}^H(\cdot)}(\mathsf{cma}, PK)$
  `Until` $(d = \mathsf{breakin})$ or $(j = T)$
  `If` $d \ne \mathsf{breakin}$ and $j = T$ `then` $j \leftarrow T+1$
  $(M, \langle b, \zeta\rangle) \leftarrow F^H(\mathsf{forge}, SK_j)$
  `If` $Vf_{PK}^H(M, \langle b, \zeta\rangle) = 1$ and $1 \le b < j$ and M was not queried of $Sgn_{SK_b}^H(\cdot)$ in period b
    `then return` 1 `else return` 0

Figure 2: Above are the algorithms comprising our scheme $\mathsf{FSIG}[k, l, T]$— the key generation, (secret) key update, signing and verifying algorithms respectively —followed by a description of the experiment that measures the success of a forger algorithm F attacking the forward-security of the scheme. Here H: $\{0,1\}^* \to \{0,1\}^l$ is a random-oracle hash-function available to all parties.

KEY EVOLUTION. During time period j, the signer signs using key SK_j. This key is generated at the start of period j, by applying a key update algorithm to the key SK_{j-1}. (The latter is the base secret key when $j = 1$.) The update algorithm $Upd(SK_{j-1})$ is presented in Figure 2. It squares the l points of the secret key at the previous stage to get the secret key at the next stage. Once this update is performed, the signer deletes the key SK_{j-1}. Since squaring modulo N is a one-way function when the factorization of N is unknown, it is computationally infeasible to recover SK_{j-1} from SK_j. Thus key exposure during period j will yield to an attacker the current key (and also future keys) but not past keys. We refer to $S_{i,j}$ as the *i-th component of the period j secret key.* Notice that the components of the secret key for period j are related to those of the base key as follows:

$$(S_{1,j}, \ldots, S_{l,j}) \;=\; (S_{1,0}^{2^j}, \ldots, S_{l,0}^{2^j}) \; . \tag{1}$$

The length of a time period (during which a specific key is in use) is assumed to be globally known. The choice depends on the application and desired level of protection against key exposure; it could vary from seconds to days.

SIGNING. Signature generation during period j is done via the algorithm $Sgn^H_{SK_j}(M)$ described in Figure 2. It as input the secret key SK_j of the current period, the message M to be signed, and the value j of the period itself, to return a signature $\langle j, (Y, Z)\rangle$, where Y, Z are values in Z_N^*. The signer first generates the "committment" Y, and then hashes Y and the message M via a public hash function H: $\{0,1\}^* \to \{0,1\}^l$ to get an l-bit "challenge" $c = c_1 \ldots c_l$ which is viewed as selecting a subset of the components of the public key. The signer then computes a 2^{T+1-j}-th root of the product of the public key components in the selected subset, and multiplies this by a 2^{T+1-j}-th root of Y to get the value Z.

Thus, in the first time period, the signer is computing 2^T-th roots; in the second time period, 2^{T-1}-th roots; and so on, until the last time period, where it is simply computing square roots. Notice that in the last time period, we simply have the Fiat-Shamir signature scheme. (The components of the secret key SK_T are square roots of the corresponding components of the public key.) The hash function H: $\{0,1\}^* \to \{0,1\}^l$ will be assumed in the security analysis to be a random oracle.

Notice that the index j of the period during which the signature was generated is part of the signature itself. This provides some sort of "time-stamp", or claimed time-stamp, and is a crucial element of the scheme and model, since security will pertain to the ability to generate signatures having such "time-stamps" with value earlier than the current date.

VERIFYING. Verification of a candidate signature $\langle j, (Y, Z)\rangle$ for a given message M with respect to a given public key PK is performed via the $Vf^H_{PK}(M, \langle j, (Y, Z)\rangle)$ procedure of Figure 2. Note the verification process depends on the claimed "time-stamp" or period indicator j in the signature, meaning that the period j too is authenticated.

FULL SCHEME. The signature scheme is fully specified by the above four algorithms. We let $\mathsf{FSIG}[k, l, T]$ denote our scheme when the parameters are fixed as indicated.

COST. The scheme FSIG is quite efficient. Signing in period j takes $T + 1 - j + l/2$ multiplications modulo N on the average. (So the worst case cost does not exceed $T + 1 + l$ multiplications, and in fact the scheme gets faster as time progresses.) For typical values of the parameters, this can be less than the cost of a single exponentiation, making signing cheaper than in RSA based schemes. Verification has the same cost as signing. (We ignore the cost of hashing, which is lower than that of the modular arithmatic.)

Like the Fiat-Shamir scheme, however, the key sizes are relatively large, being proportional to l.

The size of the public key can be reduced by having its components $U_1, \ldots, U_l$ specified implicitly rather than explicitly, as values of a random-oracle hash funcion on some fixed points. That is, the signer can choose some random constant U, say 128 bits long, and then specify small values $a_1, \ldots, a_l$ such that $H^*(U, a_i)$ is a square modulo N, where H^* is a hash function with range Z_N^*. The public key is now $(N, T, U, a_1, \ldots, a_l)$. Since N is a Blum-Williams integer, the signer can then compute a 2^{T+1}-th root of $u_i = H(U, a_i)$, and thereby have a secret key relating to the public key in the same way as before. The average size of the list $a_1, \ldots, a_l$ is very small, about $O(l \lg(l))$ bits. Unfortunately it is not possible to similarly shrink the size of the secret key in this scheme, but moving to post-Fiat-Shamir identification-derived signature schemes, one can get shorter keys.

Validity of genuine signatures. Before we discuss security, we should check that signatures generated by the signing process will always be accepted by the verification process.

Proposition 4.1 Let $PK = (N, T, U_1, \ldots, U_l)$ and $SK_0 = (N, T, 0, S_{1,0}, \ldots, S_{l,0})$ be a key pair generated by the above key generation algorithm. Let $\langle j, (Y, Z) \rangle$ be an output of $Sgn_{PK}^H(M)$. Then $Vf_{PK}(M, \langle j, (Y, Z) \rangle) = 1$.

Proof: Let $c_1 \ldots c_l \leftarrow H(j, Y, M)$. We check that $Z^{2^{(T+1-j)}} = Y \cdot \prod_{i=1}^l U_i^{c_i} \bmod N$ using Equation (1) and the fact that the signature was correctly generated:

$$\begin{aligned} Z^{2^{(T+1-j)}} &= \left(R \cdot \textstyle\prod_{i=1}^l S_{i,j}^{c_i}\right)^{2^{(T+1-j)}} \bmod N \\ &= R^{2^{(T+1-j)}} \cdot \textstyle\prod_{i=1}^l S_{i,0}^{c_i \cdot 2^{j+(T+1-j)}} \bmod N \\ &= Y \cdot \textstyle\prod_{i=1}^l S_{i,0}^{c_i \cdot 2^{(T+1)}} \bmod N \\ &= Y \cdot \textstyle\prod_{i=1}^l U_i^{c_i} \bmod N \,, \end{aligned}$$

as desired. ▮

4.2 Security of our scheme

Security model. Security is evaluated using the model of Section 2, augmented to take into account the presence of the random oracle H. Namely for any adversary F attacking the forward-security of our signature scheme using a chosen-message attack, we consider the experiment measuring its success, and thereby associate to F a success probability. For clarity we have reproduced in Figure 2 the experiment measuring the success of the adversary in the random oracle model. (In reading the experiment, recall that SK_{T+1} is the empty string by definition.) We suggest that the reader refer back to Section 2 for explanations. We now summarize the definitions that extend those of Section 2 to the random oracle model. We begin by letting $\mathbf{Succ}^{\text{fwsig}}(\mathsf{FSIG}[k, l, T], F)$ denote the probability that the experiment F-Forge$(\mathsf{FSIG}[k, l, T], F)$ returns 1. (This is the probability that the adversary F is successful in breaking FSIG in the forward security sense.) Then the insecurity function of our scheme is defined as

$$\mathbf{InSec}^{\text{fwsig}}(\mathsf{FSIG}[k, l, T]; t, q_{\text{sig}}, q_{\text{hash}}) \;=\; \max_F \{ \, \mathbf{Succ}^{\text{fwsig}}(\mathsf{FSIG}[k, l, T], F) \, \} \,.$$

The maximum here is over all F for which the following are true; the execution time of the above experiment, plus the size of the code of F, is at most t; F makes a total of at most q_{sig} queries to the signing oracles across all the stages; the total number of queries made to H in the execution of the experiment is at most q_{hash}. Note the execution time is not just that of F but rather that of the entire experiment F-Forge$(\mathsf{FSIG}[k, l, T], F)$, so includes in particular the time to compute answers to oracle queries. The time to pick the hash function H is also included, measured dynamically by counting the time to reply to each hash oracle query by picking the response randomly at the time the query is made. Similarly, q_{hash} is the number of H queries in the experiment, not just the number made explicitly by F, so that it includes the H queries made by the signing and verifying algorithms. In particular, q_{hash} is always at least one due to the hash query made in the verification of the forgery.

The smaller the value taken by the insecurity function, the more secure the scheme. Our goal will accordingly be to upper bound the values taken by this insecurity function.

FACTORING. We will prove the security of our scheme by upper bounding its insecurity as a function of the probability of being able to factor the underlying modulus in time related to the insecurity parameters. To capture this, let $Fct(\cdot)$ be any algorithm that on input a number N, product of two primes, attempts to return its prime factors, and consider the experiment:

Experiment Factor(k, Fct)
Pick random, distinct $k/2$ bit primes p, q, each congruent to 3 mod 4
$N \leftarrow pq$
$p', q' \leftarrow Fct(N)$
If $p'q' = N$ and $p' \neq 1$ and $q' \neq 1$ `then return` 1 `else return` 0

Let $\mathbf{Succ}^{\mathrm{fac}}(Fct, k)$ denote the probability that the above experiment returns 1. Let $\mathbf{InSec}^{\mathrm{fac}}(k, t)$ denote the maximum value of $\mathbf{Succ}^{\mathrm{fac}}(k, Fct)$ over all algorithms Fct for which the running time of the above experiment plus the size of the description of Fct is at most t. This represents the maximum probability of being able to factor a k-bit Blum-Williams integer in time t. We assume factoring is hard, meaning $\mathbf{InSec}^{\mathrm{fac}}(k, t)$ is very low as long as t is below the running time of the best known factoring algorithm, namely about $2^{1.9k^{1/3}\lg(k)^{2/3}}$.

SECURITY OF OUR SIGNATURE SCHEME. We are able to prove that as long as the problem of factoring Blum-Williams integers is computationally intractable, it is computationally infeasible to break the forward security of our signature scheme. The following theorem provides a precise, quantitative statement, upper bounding the forward insecurity of our scheme as a function of the insecurity of factoring.

Theorem 4.2 Let $\mathsf{FSIG}[k, l, T]$ represent our key evolving signature scheme with parameters modulus size k, hash function output length l, and number of time periods T. Then for any t, any q_{sig}, and any $q_{\mathrm{hash}} \geq 1$

$$\begin{aligned}&\mathbf{InSec}^{\mathrm{fwsig}}(\mathsf{FSIG}[k, l, T]; t, q_{\mathrm{sig}}, q_{\mathrm{hash}})\\ &\quad\leq\ q_{\mathrm{hash}} \cdot (T+1) \cdot \left[2^{-l} + \sqrt{2lT \cdot \mathbf{InSec}^{\mathrm{fac}}(k, t')}\right] + \frac{4q_{\mathrm{sig}} \cdot q_{\mathrm{hash}}}{2^k}\ ,\end{aligned}$$

where $t' = 2t + O(k^3 + k^2 l \lg(T))$.

The security parameter k must be chosen large enough that the $\mathbf{InSec}^{\mathrm{fac}}(k, t')$ (the probability of being able to factor the modulus in time t') is very low. The theorem then tells us that the probability of being able to compromise the forward security of the signature scheme is also low.

The theorem above proves the security of the scheme in a qualitative sense: certainly it implies that polynomial time adversaries have negligible advantage, which is the usual complexity based goal. However it also provides the concrete, or exact security, via the concrete indicated bound. In this case however, we know no attacks achieving a success matching our bound, and suspect our bound is not tight. Perhaps the concrete security can be improved, particularly with regard to the $q_{\text{hash}} \cdot (T+1)$ multiplicative factor.

The proof of this theorem is in two steps. Section 5 makes explicit an identification scheme that underlies the above signature scheme, and Lemma 5.1 shows that this identification scheme is forward secure as long as factoring is hard. Section 6 relates the forward security of our signature scheme to that of the identification scheme via Lemma 6.1. Putting these two lemmas together easily yields Theorem 4.2. We now turn to the lemmas.

5 Our forward-secure identification scheme

We first discuss the problem of forward-secure identification. Then we specify our scheme. Finally we analyze the security of our scheme.

5.1 Forward-secure identification

We consider the standard framework for a public-key based identification protocol. The prover is in possession of secret key SK_0, while both the prover and the verifier are in possession of the corresponding public key PK. The prover wants to identify herself interactively to the verifier. A standard three-pass protocol will be used, consisting of a "commitment" Y from the prover, followed by a challenge c from the verifier, and finally an "answer" Z from the prover. The standard security concern is that an adversary (not in possession of the secret key) be able to identify itself to the prover under the public key PK of the legitimate prover.

We extend the setting to allow evolution of the secret key, enabling us to consider forward security. Thus the time over which the public key is valid is divided into T periods, and in period j the prover identifies itself using a secret key SK_j. The public key remains fixed, but the protocol and verification procedure depend on j, which takes the value of the current period, a value that all parties are agreed upon. Forward security means that an adversary in possession of SK_j should still find it computationally infeasible to identify itself to the verifier in a time period i previous to j.

This security measure having been stated should, however, at once create some puzzlement. Identification is an "on-line" activity. More specifically, verification is an on-line, one-time action. Once period i is over, there is no practical meaning to the idea of identifying oneself in period i; one cannot go back in time. So forward security is not a real security concern or attribute in identification. So why consider it? For us, forward-secure identification is only a convenient mathematical game or abstraction based on which we can analyze the forward security of our signature scheme. (And forward security of the signature scheme is certainly a real concern: the difference with identification is that for signatures verification can take place by anyone at any time after the signature is created.)

In other words, we can certainly define and consider a mathematically (if not practically) meaningful notion of forward security of an identification scheme, and analyze a scheme with regard to meeting this property. That is what we do here, for the purpose of proving Theorem 4.2.

5.2 Our scheme

Keys (both public and secret) are identical to those of our signature scheme, and are generated by the same procedure KG that we described in Section 4, so that at the start of the game the prover is in possession of the base secret key SK_0. The public key PK is assumed to be in possession of the verifier. At the start of period j $(1 \leq j \leq T)$ the prover begins by updating the previous secret key SK_{j-1} to the new secret key SK_j that she will use in period j. This process is done by the same update algorithm as for the signature scheme, specified in Section 4. The update having been performed, the key SK_{j-1} is deleted. During period j, the prover and verifier may engage in any number of identification protocols with each other, as need dictates. In these protocols, it is assumed both parties are aware of the value j indicating the current period. The prover uses key SK_j to identify herself. We depict in Figure 3 the identification protocol in period j, showing the steps taken by both the prover and the verifier.

The identification scheme is fully specified by key generation algorithm, key update algorithm, and the proving and verifying algorithms that underly Figure 3. We let $\mathsf{FID}[k,l,T]$ denote our identification scheme when the parameters are fixed as indicated.

It should of course be the case that when the prover is honest, the verifier accepts. The proof of this is just like the proof of Proposition 4.1, the analogous issue for the signature scheme.

As should be clear from the identification protocol of Figure 3, the signing procedure under key SK_j that we defined in Section 4 is closely related to the identification protocol in period j: the former is derived from the latter by the standard process of specifying the challenge c via the value of a hash function on the message M and the commitment Y. This connection enables us to break the security analysis of the signature scheme into two parts; a forward-security analysis of the identification scheme, and an analysis of the preservation of forward-security of the "challenge hash" paradigm. This section is devoted to the first issue. We begin with the model.

5.3 Forward-security of our identification scheme

Security model. The adversary in an identification scheme is called an impersonator and is denoted I. It knows the public key PK of the legitimate prover, and the current period j. We view I as functioning in two stages: the break-in phase (breakin) and the impersonation phase (imp). Its success probability in breaking FID is determined by the experiment F-Impersonate($\mathsf{FID}[k,l,T], I$) described in Figure 3. In the break-in phase the adversary returns a time period $b \in \{1,\ldots,T\}$ as a break-in time, as a function of the public key. It will be returned the corresponding secret key SK_b, and now will try to successfully identify itself relative to some period $j < b$ of its choice. Here it will try to impersonate the prover, so first chooses some commitment value Y. Upon being given the random challenge $c_1 \ldots c_l$ from the verifier, it returns an answer Z. It is considered successful if the verifier accepts the answer. Above, U_i is the i-th component of the public key PK. We let $\mathbf{Succ}^{\mathrm{fwid}}(\mathsf{FID}[k,l,T], I)$ denote the probability that the above experiment returns 1 and then let

$$\mathbf{InSec}^{\mathrm{fwid}}(\mathsf{FID}[k,l,T]; t) \;=\; \max_I \{\, \mathbf{Succ}^{\mathrm{fwid}}(\mathsf{FID}[k,l,T], I) \,\} \;,$$

the maximum being over all I that for which the execution time of the above experiment, plus the size of the code of F, is at most t. As usual, the smaller the value taken by the insecurity function, the more secure the scheme. Our goal will be to upper bound the values taken by this insecurity function.

Notice that our model does not allow the adversary to first play the role of a cheating verifier, as does the standard model of identification. The reason is that we are only interested in applying this to the signature scheme, and there the challenge, being the output of the hash function, will be random, corresponding to an honest verifier. So it suffices to consider an honest verifier here.

Security result. Next, we prove the security of our identification scheme by showing that breaking the forward security of our identification scheme is hard as long as the problem of factoring a Blum-Williams integer into its two prime factors is hard. The following lemma states a result which implies this:

Prover		**Verifier**
Has : $SK_j = (N, T, j, S_{1,j}, \ldots, S_{l,j})$		Has : $PK = (N, T, U_1, \ldots, U_l)$ and j
$R \stackrel{R}{\leftarrow} Z_N^* \text{ ; } Y \leftarrow R^{2^{(T+1-j)}} \bmod N$		
	$\xrightarrow{\quad Y \quad}$	
		$c_1 \ldots c_l \stackrel{R}{\leftarrow} \{0,1\}^l$
	$\xleftarrow{\quad c_1 \ldots c_l \quad}$	
$Z \leftarrow R \cdot \prod_{i=1}^{l} S_{i,j}^{c_i} \bmod N$		
	$\xrightarrow{\quad Z \quad}$	
		`If` $Z^{2^{(T+1-j)}} = Y \cdot \prod_{i=1}^{l} U_i^{c_i} \bmod N$
		`then` accept `else` reject

Experiment F-Impersonate($\mathsf{FID}[k, l, T], I$)

$(PK, SK_0) \stackrel{R}{\leftarrow} KG(k, l, T)$

$b \leftarrow I(\mathsf{breakin}, PK)$ [$1 \leq b \leq T + 1$ is the break-in time.]

`For` $j = 1, \ldots, b$ `do` $SK_j \leftarrow Upd(SK_{j-1})$; $j \leftarrow j + 1$ `EndFor`

$Y, j \leftarrow I(\mathsf{imp}, SK_b)$ [$1 \leq j < b$ is the impersonation time.]

$c_1 \ldots c_l \stackrel{R}{\leftarrow} \{0,1\}^l$ [Challenge from verifier]

$Z \leftarrow I(c_1 \ldots c_l)$ [Response of impersonator-prover]

`If` $Z^{2^{(T+1-j)}} = Y \cdot \prod_{i=1}^{l} U_i^{c_i} \bmod N$ and $j < b$

`then return` 1 `else return` 0

Figure 3: Above is the protocol executed in period j in our forward-secure ID scheme $\mathsf{FID}[k, l, T]$, followed by the description of the experiment that measures the success of an impersonation algorithm I attacking the forward security of the ID scheme. In the protocol, the prover has the secret key SK_j for period j and the verifier has the base public key PK and the value of j. The key generation and (secret) key update algorithms for the ID scheme are the same as those in Figure 2.

Lemma 5.1 Let $\mathsf{FID}[k, l, T]$ represent our key evolving identification scheme with modulus size k, challenge length l, and number of time periods T. Then for any t

$$\mathbf{InSec}^{\mathrm{fwid}}(\mathsf{FID}[k, l, T]; t) \quad \leq \quad 2^{-l} + \sqrt{2lT \cdot \mathbf{InSec}^{\mathrm{fac}}(k, t')} \ ,$$

where $t' = 2t + O(k^3 + k^2 l \lg(T))$.

The first term in the bound represents the probability that the impersonator guesses the verifier's challenge, in which case it can of course succeed. The second term rules out any other attacks that are very different from simply trying to factor the modulus.

5.4 Proof of Lemma 5.1

Let I be an impersonating algorithm attacking the forward security of our identification scheme in the manner described in Section 5, and assume the running time of F-Impersonate($\mathsf{FID}[k, l, T], I$) plus the size of the description of I is at most t. We will specify a factoring algorithm Fct whose success in factoring its k-bit input modulus is related to the success of I in breaking the identification scheme via

$$\mathbf{Succ}^{\mathrm{fwid}}(\mathsf{FID}[k, l, T], I) \quad \leq \quad 2^{-l} + \sqrt{2lT \cdot \mathbf{Succ}^{\mathrm{fac}}(k, Fct)} \ . \tag{2}$$

Furthermore, the running time of experiment Factor(k, Fct) plus the size of the description of Fct will be at most t'. Lemma 5.1 follows by the definitions of the insecurity functions. The rest of this proof is devoted to specifying and analyzing the algorithm Fct. Before providing the specification let us try to sketch the idea.

The modulus N is factored via the standard process of finding two different square roots of some square modulo N. Our algorithm begins by picking a random square v in Z_N^*, of which it knows one square root and will seek another. It creates a public key on which to run the impersonator, in such a way that $S_{i,b'}^2 = v \bmod N$ where $i \in \{1, \ldots, l\}$ and $b' \in \{1, \ldots, T\}$ are randomly chosen. Remember that $S_{i,b'}$ is the i-th component of the period b' secret key $SK_{b'}$. Our factoring algorithm does not a priori know the value of $S_{i,b'}$ (it will use I to find it) but this value is implicitly defined once the public key is fixed, as that $2^{T+1-b'}$-th root of U_i (the i-th component in the public key) which is itself a square, this value being unique because the modulus is a Blum-Williams integer. Even without knowing $S_{i,b'}$, however, our algorithm can arrange that $S_{i,b'}^2 = v \bmod N$ by an appropriate choice of U_i depending on v. Now the impersonator I, when run on the public key, returns some break-in time $b \in \{1, \ldots, T+1\}$. Our factoring algorithm hopes that $b > b'$ (which happens with high enough probability) because then it is capable of returning to I the secret key SK_b of stage b that I requested, in a manner which we will detail later. When given the key, I starts to play the role of the prover, identifying itself with respect to some period $a < b$ of its choice. As long as $a \leq b'$ it will be possible to extract a $2^{b'+1-a}$-th root (square root if $a = b'$, fourth root if $a = b' - 1$, etc.) of v by an "extraction" process of running the impersonator on two different challenge vectors, as in a typical "proof of knowledge" setting.

The factoring algorithm Fct is described in detail in Figure 4. It is understood that when this algorithm runs I as a subroutine, it picks at the beginning a sequence of random coins ρ for I and leaves them fixed throughout the execution. It also preserves the state of I across invocations of I made in a single execution.

We now analyze this algorithm towards establishing Equation (2). First, we need a technical lemma about solvability of equations in Z_N^*. As an illustration, a simple case is that we have $\alpha^{2^n} = \alpha_1^{2^{n_1}} \bmod N$ where $\alpha_1, \alpha \in Z_N^*$, α is a square, and $n_1 > n \geq 0$. We would like to "solve" for

`Algorithm` $Fct(N)$

1⟩ $w \stackrel{R}{\leftarrow} Z_N^*$; $v \leftarrow w^2 \bmod N$ [Will try to find a square root of v that is different from w.]

2⟩ $i \stackrel{R}{\leftarrow} \{1, \ldots, l\}$ [Choose a component in the key at which to set a "trap".]

3⟩ $S_1, \ldots, S_{i-1}, S_{i+1}, \ldots, S_l \stackrel{R}{\leftarrow} Z_N^*$ [Choose all but the i-th component of the secret key just like the key generation algorithm would.]

4⟩ $b' \stackrel{R}{\leftarrow} \{1, \ldots, T\}$ [Choose a time period at which to set a "trap".]

5⟩ `For` $m = 1, \ldots, l$ `do` `If` $m = i$ `then` $U_m \leftarrow v^{2^{T-b'}}$ `else` $U_m \leftarrow S_m^{2^{(T+1)}} \bmod N$ `EndIf` `EndFor` [The intention is that $S_{i,b'}^2 = v \bmod N$. Of course, our algorithm does not know the values $S_{i,n}$ for $n \leq b'$. Indeed, the point is to use the impersonator to find one of them.]

6⟩ $PK \leftarrow (N, T, U_1, \ldots, U_l)$ [Create a public key for the ID scheme.]

7⟩ $b \leftarrow I(\mathsf{breakin}, PK)$ [Run the impersonator to get a break-in time $b \in \{1, \ldots, T+1\}$.]

8⟩ `If` $b \leq b'$ `then abort else` continue [We would not be able to supply SK_b to I.]

9⟩ `If` $b = T+1$ `then` let SK_b be the empty string and goto Step `11` `else` continue

10⟩ `For` $m = 1, \ldots, l$ `do` `If` $m = i$ `then` $S_{m,b} \leftarrow v^{2^{b-b'-1}}$ `else` $S_{m,b} \leftarrow S_m^{2^b} \bmod N$ `EndIf` `EndFor` $SK_b \leftarrow (N, T, b, S_{1,b}, \ldots, S_{l,b})$ [It must be that $S_{i,b} = S_{i,b'}^{2^{b-b'}}$. Our intention is to make it look like $S_{i,c}^2 = v$, so this means we must set $S_{i,b} = v^{2^{b-b'-1}}$.]

11⟩ $Y, a \leftarrow I(\mathsf{imp}, SK_b)$ [Give the impersonator the secret key for period b and obtain an impersonation time $a < b$ and the first move Y for the "prover" for the ID protocol.]

12⟩ `If` $a > b'$ `then abort` [We would not be able to use a successful impersonation to get a square root of v so we abort.]

13⟩ $c_{1,1} \ldots c_{l,1} \stackrel{R}{\leftarrow} \{0,1\}^l$; $c_{1,2} \ldots c_{l,2} \stackrel{R}{\leftarrow} \{0,1\}^l$ [Choose two random challenge vectors, each in the manner the verifier would choose a challenge.]

14⟩ $Z_1 \leftarrow I(c_{1,1} \ldots c_{l,1})$; $Z_2 \leftarrow I(c_{1,2} \ldots c_{l,2})$ [Provide $c_{1,1} \ldots c_{l,1}$ to the impersonator-prover as the challenge to Y and get the corresponding reply Z_1, and then do the same for $c_{1,2} \ldots c_{l,2}$, by "backing up" I.]

15⟩ `If` $c_{i,1} = c_{i,2}$ `then` abort `else` continue [If the challenge vectors agree in the "trap" position we won't win.]

16⟩ $S_{i,a} \leftarrow \left[\frac{Z_1}{Z_2} \cdot \prod_{m \neq i} S_m^{(c_{m,2}-c_{m,1})2^a}\right]^{c_{i,1}-c_{i,2}} \bmod N.$ [Notice we use the values S_m only for $m \neq i$; indeed we do not even know S_i.]

17⟩ $S_{i,b'} \leftarrow S_{i,a}^{2^{b'-a}} \bmod N$; $x \leftarrow S_{i,b'}$

18⟩ `If` $x \equiv \pm w \bmod N$ `then abort else` continue

19⟩ $h \leftarrow \gcd(w - x, N)$ [A non-trivial factor of N.] `Return` $(h, N/h)$ [The factorization of N]

Figure 4: Factoring algorithm that uses a given impersonating algorithm I as a subroutine to factor a given Blum-Williams integer N.

α as $\alpha = \alpha_1^{2^{n_1-n}}$. In general this will not be correct, because square roots are not unique in Z_N^*. However, it is correct when N is Blum-Williams integer, because then each square has a unique square root which is itself a square. The following lemma states a more general version of this.

Lemma 5.2 Suppose N is a Blum-Williams integer. Suppose $\alpha, \alpha_1, \ldots, \alpha_t \in Z_N^*$ and α is a square modulo N. Suppose $n, n_1, \ldots, n_t$ are integers such that $n_1, \ldots, n_t > n \geq 0$. Suppose

$$\alpha^{2^n} \;=\; \prod_{j=1}^{t} \alpha_j^{2^{n_j}} \bmod N \,. \tag{3}$$

Then

$$\alpha \;=\; \prod_{j=1}^{t} \alpha_j^{2^{n_j-n}} \bmod N \,.$$

Proof: Let S be the set of squares in Z_N^* and $f\colon\, S \to S$ the map $f(x) = x^2 \bmod N$. We know that f is a permutation on S because N is a Blum-Williams integer [6, 22]. Let $f^{-1}\colon\, S \to S$ be the inverse map of f. Each number $u = x^2 \in S$ has exactly four square roots, precisely one of which, namely x, is in S. We know that f^{-1} is the inverse of a permutation, so $f^{-1}(u)$ must itself be a square for any $u \in S$. This means that $f^{-1}(u) = x$, and none of the other three square roots of u. Now, α^{2^n} is given as a product of factors α_j each raised to some power of the form 2^{n_j}. So we know that decreasing by one the n_j in each exponent in the product will result in a square root of α^{2^n}, due to the fact that each $n_j > n$. But because each resulting exponent will also be of the form 2^i, we know that this resulting product will also be a square. This means that we have found the desired square root, x. Repeatedly applying this process n times will indeed give us the desired value α. ▮

Claim 5.3 Consider an execution of $Fct(N)$ in which the algorithm does not abort, and also the impersonator succeeds under both the given challenges, meaning

$$Z_d^{2^{T+1-a}} = Y \cdot \prod_{m=1}^{l} U_m^{c_{m,d}} \quad \text{for both } d = 1 \text{ and } d = 2.$$

Then the quantity x computed by $Fct(N)$ is a square root of v.

Proof: We need to check that $x^2 = v \bmod N$. In this proof, we assume all computations are performed modulo N. We let a represent the impersonation time, as in the $Fct(N)$ algorithm. Note that $1 \leq a \leq b' < b \leq T+1$ since the algorithm did not abort. Since the impersonator succeeds under both challenges, we know the following two equations must hold:

$$Z_1^{2^{T+1-a}} = Y \cdot \prod_{m=1}^{l} U_m^{c_{m,1}} \quad \text{and} \quad Z_2^{2^{T+1-a}} = Y \cdot \prod_{m=1}^{l} U_m^{c_{m,2}} \,.$$

Dividing, we get

$$\left(\frac{Z_1}{Z_2}\right)^{2^{T+1-a}} = \prod_{m=1}^{l} U_m^{c_{m,1}-c_{m,2}} \,.$$

But from the code of $Fct(N)$ we know that $U_i = v^{2^{T-b'}}$. So we can write

$$\left(\frac{Z_1}{Z_2}\right)^{2^{T+1-a}} = \left(v^{2^{T-b'}}\right)^{(c_{i,1}-c_{i,2})} \cdot \prod_{m \neq i} U_m^{c_{m,1}-c_{m,2}} \,.$$

Re-arrange terms to get

$$\left(v^{2^{T-b'}}\right)^{(c_{i,1}-c_{i,2})} \;=\; \left(\frac{Z_1}{Z_2}\right)^{2^{T+1-a}} \cdot \prod_{m \neq i} U_m^{c_{m,2}-c_{m,1}}$$

$$= \left(\frac{Z_1}{Z_2}\right)^{2^{T+1-a}} \cdot \prod_{m \neq i} S_m^{(c_{m,2}-c_{m,1})2^{T+1}}$$

We would like to "solve" this equation for v. A natural thought is to raise both sides to the power $2^{-(T-b')}$, but this operation does not make sense, since $2^{-(T-b')}$ does not even have an inverse modulo $\varphi(N)$. Indeed, in general, the solution for v is not unique. However, we know that N is a Blum-Williams integer, so we can apply Lemma 5.2 with $\alpha = v^{c_{i,1}-c_{i,2}}$ and $n = T - b'$. Note that α is a square (because v is a square), and also $T+1-a > T-b' \geq 0$ (because $1 \leq a \leq b' < T+1$). Thus we get

$$\begin{aligned} v^{c_{i,1}-c_{i,2}} &= \left(\frac{Z_1}{Z_2}\right)^{2^{(T+1-a)-(T-b')}} \prod_{m \neq i} S_m^{(c_{m,2}-c_{m,1})2^{(T+1)-(T-b')}} \\ &= \left(\frac{Z_1}{Z_2}\right)^{2^{b'+1-a}} \prod_{m \neq i} S_m^{(c_{m,2}-c_{m,1})2^{b'+1}} \\ &= \left[\frac{Z_1}{Z_2} \cdot \prod_{m \neq i} S_m^{(c_{m,2}-c_{m,1})2^{a}}\right]^{2^{b'-a+1}} . \end{aligned}$$

Note $c_{i,1} - c_{i,2} \in \{-1, +1\}$ (since otherwise the algorithm would have aborted). Thus we can raise both sides of the above to this power and get

$$v = \left[\frac{Z_1}{Z_2} \cdot \prod_{m \neq i} S_m^{(c_{m,2}-c_{m,1})2^{a}}\right]^{(c_{i,1}-c_{i,2})2^{b'-a+1}} = S_{i,a}^{2^{b'-a+1}} = S_{i,b'}^2 = x^2 ,$$

the last equalities being from the definitions of $S_{i,a}, S_{i,b'}$ and x in the algorithm. We therefore have shown that $x^2 = v \bmod N$, as desired. ▌

Now we want to lower bound the probability of the event considered in the previous lemma. One-half of this probability is a lower bound on the probability that $Fct(N)$ factors N because $x \neq w \bmod N$ with probability $1/2$.

Success of the factoring algorithm requires that the impersonator succeed on two challenges. The probability of this is however not just the square of the overall success probability of the impersonator, because the two successes are related events, both relating to the same choice of Y, a for the impersonator. Our analysis proceeds by considering the success probability under a fixed public key PK and sequence of coins ρ for I, and then averaging this appropriately. Accordingly define $\mathrm{Succ}(PK, \rho)$ as the probability that I succeeds in breaking the forward security of the identification scheme when the public key is fixed to PK and the coins of I are fixed to ρ. (The probability here is solely over the choice of the l-bit challenge vector.) By definition, the overall success probability of I is

$$\mathbf{Succ}^{\mathrm{fwid}}(\mathsf{FID}[k,l,T], I) \;=\; \mathbf{E}\left[\mathrm{Succ}(PK, \rho)\right] , \tag{4}$$

the expectation being over the choices of PK and ρ, the first chosen as per the key generation algorithm, and the second at random. The following claim lower bounds the probability that Fct succeeds in factoring its input N in terms of the expectation of a quantity involving $\mathrm{Succ}(PK, \rho)$.

Claim 5.4 For any modulus size k we have

$$\mathbf{Succ}^{\mathrm{fac}}(k, Fct) \ \geq\ \mathbf{E}\left[\frac{\mathrm{Succ}(PK,\rho)\left(\mathrm{Succ}(PK,\rho) - 2^{-l}\right)}{2lT}\right],$$

the expectation being over the choices of PK and ρ, the first selected according to the key generation algorithm KG and the second selected at random.

Proof: From Claim 5.3 and the code of Fct we see that the conditions under which $Fct(N)$ succeeds in factoring N are that $1 \leq a \leq b' < b \leq T+1$, the impersonator succeeds on both the random challenge vectors $c_{1,1} \ldots c_{l,1}$ and $c_{1,2} \ldots c_{l,2}$, these vectors disagree in the i-th component, and the final value of x is different from $\pm w \bmod N$. We want to lower bound the probability of all this happening in the experiment $\mathrm{Factor}(k, Fct)$.

It is important to first note here that the claimed lower bound is a function of quantities referring to the "real" experiment $\text{F-Impersonate}(\mathsf{FID}[k,l,T], I)$ in which I takes as input a public key generated by KG, provides a break-in time, and then interacts with the real verifier. The event describing the success of Fct however concerns the execution of I as a subroutine within Fct. To relate this, we begin by claiming that in the execution of I within Fct, the "view" of I is the same as in the real game as long as Fct did not abort. Let us briefly verify this. The issue is the distribution of any inputs fed to I within Fct. The first such input is the public key; the second is the two choices of challenge vectors.

First, consider the distribution of the public key that is the input to I. We claim that for both the "real' public key generated according to the key generation algorithm, and for the one generated within Fct, the components of the public key are all randomly and independently distributed squares mod N. This is true because N is a Blum-Williams integer, so that squaring mod N is a permutation on the set of squares in Z_N^*. In the real experiment, each U_m is the result of raising a randomly chosen S_m to the power 2^{T+1}, resulting in a random square. Clearly any S_m where $m \neq i$ that is created in the Fct algorithm has the same distribution since it was created the same way. The only concern then is with the creation of U_i. This is set in line 5 to the $2^{T-b'}$-th power of v, and hence is a square since v is a square and $T - b' \geq 0$. Secondly, both challenge vectors inside the Fct algorithm are chosen at random at line 13, each distributed like a real challenge vector. This verifies the claim about the "view" of I subject when it is run by Fct, conditional to not aborting the execution.

Henceforth we may assume the public key PK is chosen according to the distribution of the key generation algorithm.

Now fix a public key PK and coins ρ for I. If the two chosen challenge vectors differ in at least one position then the random choice of i in line 2 yields a probability of at least $1/l$ that $c_{i,1} \neq c_{i,2}$, meaning the factoring algorithm does not abort in line 15. That probability that I succeeds on two random but distinct challenge vectors is $\mathrm{Succ}(PK,\rho)\left(\mathrm{Succ}(PK,\rho) - 2^{-l}\right)$, so dividing this by l yields the probability of success conditional on not aborting for any other reason. The other reasons for abortion occur at lines 8, 12, or 18 in the code, and we now consider the effect of this on the success probability. The probability that $1 \leq a \leq b' < b \leq T+1$ is at least $1/T$ independently of anything else, due to the random choice of b' and the fact that $a < b$. The probability that $x \neq \pm w \bmod N$ is $1/2$ independently of anything else due to the random choice of w. This means that the success probability of the factoring algorithm when it runs I with key PK is at least

$$\frac{\mathrm{Succ}(PK,\rho)\left(\mathrm{Succ}(PK,\rho) - 2^{-l}\right)}{2lT}.$$

Now, the overall success probability of the factoring algorithm is the expectation of this over the choice of PK, ρ. But due to the above claim about the "view" of I, this expectation can be taken under the choice of PK as given by the key generation algorithm. ▮

To shorten notation let $\epsilon = \mathbf{Succ}^{\mathrm{fwid}}(\mathsf{FID}[k,l,T], I)$. We use Jensen's inequality and Equation (4) to simplify the lower bound of Claim 5.4 as follows:

$$\begin{aligned}
\delta &\stackrel{\mathrm{def}}{=} \mathbf{Succ}^{\mathrm{fac}}(k, Fct) \\
&\geq \frac{1}{2lT} \cdot \mathbf{E}\left[\mathrm{Succ}(PK,\rho)\left(\mathrm{Succ}(PK,\rho) - 2^{-l}\right)\right] \\
&\geq \frac{1}{2lT} \cdot \left(\mathbf{E}\left[\mathrm{Succ}(PK,\rho)^2\right] - 2^{-l} \cdot \mathbf{E}\left[\mathrm{Succ}(PK,\rho)\right]\right) \\
&\geq \frac{1}{2lT} \cdot \left(\mathbf{E}\left[\mathrm{Succ}(PK,\rho)\right]^2 - 2^{-l} \cdot \mathbf{E}\left[\mathrm{Succ}(PK,\rho)\right]\right) \\
&\geq \frac{1}{2lT} \cdot \left[\mathbf{Succ}^{\mathrm{fwid}}(\mathsf{FID}[k,l,T], I)^2 - 2^{-l} \cdot \mathbf{Succ}^{\mathrm{fwid}}(\mathsf{FID}[k,l,T], I)\right] \\
&= \frac{\epsilon^2 - 2^{-l}\epsilon}{2lT} \, .
\end{aligned}$$

We need to upper bound ϵ in terms of δ, l, T. We are looking at the quadratic inequality

$$f(\epsilon) \stackrel{\mathrm{def}}{=} \epsilon^2 - 2^{-l}\epsilon - 2lT\delta \;\leq\; 0 \, .$$

As a function of ϵ, the function f is first decreasing, then increasing. It has two roots, the smaller of which is negative. Thus if we denote by ϵ_0 the value of the larger (positive) root of f, the upper bound is $\epsilon \leq \epsilon_0$. So now we need to upper bound ϵ_0 as a function of δ, l, T. By the quadratic formula we have

$$\begin{aligned}
\epsilon_0 &= \frac{2^{-l} + \sqrt{2^{-2l} + 8lT\delta}}{2} \\
&\leq \frac{2^{-l}}{2} + \frac{\sqrt{2^{-2l}}}{2} + \frac{\sqrt{8lT\delta}}{2} \\
&= 2^{-l} + \sqrt{2lT\delta} \, .
\end{aligned}$$

This establishes Equation (2) and completes the proof of Lemma 5.1.

The purpose of the quadratic equation manipulation above was to make the first term of the bound 2^{-l}, rather than the $2^{-l/2}$ that would have arisen had we simplified by using 2^{-l} as an upper bound on $2^{-l}\epsilon$ and thereby avoided having a term linear in ϵ in the above.

6 From identification to signatures

As we have noted, our signature scheme is derived from our identification scheme in the standard way, namely by having the signer specify the challenge c as a hash of the message M and commitment Y. In the standard setting we know that this paradigm works, in the sense that if the identification scheme is secure then so is the signature scheme [18, 16]. However we are not in the standard setting; we are considering an additional security property, namely forward security. The previous results do not address this. It turns out however that the hashing paradigm continues to work in the presence of forward security, in the following sense: if the identification scheme is forward secure, so is the derived signature scheme. We first state the result saying this and then prove it.

6.1 Preservation of forward-security

The following lemma says that if our identification scheme is forward-secure then so is our signature scheme. Moreover it provides the concrete security analysis of the reduction.

Lemma 6.1 Let $\mathsf{FID}[k,l,T]$ and $\mathsf{FSIG}[k,l,T]$ represent, respectively our key evolving identification scheme and our key evolving signature scheme, with parameters k,l,T. Then for any t, any q_{sig}, and any $q_{\text{hash}} \geq 1$

$$\mathbf{InSec}^{\text{fwsig}}(\mathsf{FSIG}[k,l,T]; t, q_{\text{sig}}, q_{\text{hash}}) \quad \leq \quad q_{\text{hash}} \cdot (T+1) \cdot \mathbf{InSec}^{\text{fwid}}(\mathsf{FID}, t') + \frac{4q_{\text{sig}} \cdot q_{\text{hash}}}{2^k} ,$$

where $t' = t + O(k^2 l \lg(T))$.

Thus if the forward insecurity of the identification scheme is small, so is that of the signature scheme. Combining Lemmas 6.1 and 5.1 proves Theorem 4.2, the main theorem saying our signature scheme is forward-secure.

Before proving this lemma we remark that it is actually quite general. It applies to the transformation of most "standard" ID schemes to signature schemes. The requirement from the ID scheme is that it be a three move scheme in which the second move is a random challenge from the verifier, and that the protocol be honest-verifier zero-knowledge.

6.2 Proof of Lemma 6.1

Let F be a forger algorithm attacking the forward security of our signature scheme in the manner described in Section 4. Assume the running time of F-Forge$(\mathsf{FSIG}[k,l,T], F)$ plus the size of the description of F is at most t, that the number of sign queries made by F is at most q_{sig} and that the number of hash queries made in the experiment is at most q_{hash}. We will specify an impersonation algorithm I attacking the forward security of our identification scheme in the manner described in Section 5. The success of I in breaking the forward-security of the identification scheme will be related to the success of F in breaking the forward-security of the signature scheme via

$$\mathbf{Succ}^{\text{fwid}}(\mathsf{FID}[k,l,T], I) \quad \geq \quad \frac{\mathbf{Succ}^{\text{fwsig}}(\mathsf{FSIG}[k,l,T], F) - (q_{\text{hash}} \cdot q_{\text{sig}}) \cdot 2^{-k}}{q_{\text{hash}} \cdot (T+1)} . \tag{5}$$

Furthermore, the running time of experiment F-Impersonate$(\mathsf{FID}[k,l,T], I)$ plus the size of the description of I will be at most t'. Lemma 6.1 follows by the definitions of the insecurity functions. The rest of this proof is devoted to specifying and analyzing the algorithm I.

The algorithm is described in full in Figure 5, with the subroutines it calls upon described in Figure 6. We now discuss the ideas at a high level.

Recall that I operates in phases. The first phase is the break-in phase where it is given the public key $PK = (N, T, U_1, \ldots, U_l)$ and outputs a break-in time $b \in \{1, \ldots, T+1\}$. It obtains the secret key SK_b for this time (recall that SK_{T+1} is the empty string by default) and then must attempt to impersonate the prover in a time period $j' < b$ of its choice. For this purpose it chooses (in an impersonation phase) the value j' together with the first message Y' for the prover. It then receives a challenge $c_1 \ldots c_l$ from the verifier and must respond with an answer Z' so that the verifier accepts. The idea of our construction is that the impersonator will obtain j', Y', Z' from a forgery of the forger F. Namely let $(M, \langle j', (Y', Z') \rangle)$ denote the forgery output by F. Then the impersonator wants to output Y', j' in the impersonation phase and reply by Z' to the received challenge $c_1 \ldots c_l$. This will work as long as the forger believes that $H(j', Y', M') = c_1 \ldots c_l$. To

`Algorithm` $I(\mathsf{breakin}, PK)$ [Break-in phase: must return a break-in time period.]

1⟩ Let $b_{\text{guess}} \stackrel{R}{\leftarrow} \{1, 2, \ldots, T+1\}$ and `Return` b_{guess}

`Algorithm` $I(\mathsf{imp}, SK_{b_{\text{guess}}})$ [Impersonation phase: Given secret key $SK_{b_{\text{guess}}}$ of break-in time period, must return a time period j' for impersonation and first move (commitment) Y' for prover. Then given a challenge $c_1 \ldots c_l$ from the verifier, must return a response Z'.]

1⟩ $hcount \leftarrow 0$ [Initialize count of hash queries to zero. This will be incremented by the hash-oracle simulator] and $i_{\text{guess}} \stackrel{R}{\leftarrow} \{1, 2, ..., q_{\text{hash}}\}$ [Guess the hash query corresponding to F's attempt at forgery. This value will be used by the hash-oracle simulator.]

2⟩ Run $F(\mathsf{cma}, PK)$ [Run the forger in its chosen-message attack phase] letting $j = 1, \ldots, T$ denote the current time period during the simulation, until F chooses a break-in time or finishes its chosen-message attack in phase T. Reply to queries of F as follows:

- For any hash-oracle query (j, Y, M) made by F, execute the subroutine $H_{\text{sim}}(j, Y, M)$ of Figure 6. If this returns a value h then return h to the forger as the answer to the hash-oracle query, and continue. [It is possible the hash-oracle simulator aborts the impersonation procedure, in which case this entire impersonation algorithm fails.]
- For any sign-oracle query M made by F during time period j, execute the subroutine $Sgn_{\text{sim}}(j, M)$ of Figure 6. If this returns a value $\langle j, (Y, Z)\rangle$ then return $\langle j, (Y, Z)\rangle$ to the forger as the answer to the sign-oracle query, and continue. [It is possible the sign-oracle simulator aborts the impersonation procedure entirely, in which case this entire impersonation algorithm fails.]
- If F halts the cma phase by returning $\mathsf{breakin}$ then let j be the current period. If the cma phase for the T-th period is completed without F returning $\mathsf{breakin}$ then let $j \leftarrow T+1$. In either case `goto` Step 3. [We need to return SK_j to F in order to continue the execution of the latter, where j is the current period if F output $\mathsf{breakin}$ and $j = T+1$ if F ended the phase T without outputting $\mathsf{breakin}$.] Else continue this loop.

3⟩ `If` $j \neq b_{\text{guess}}$ `then abort else` continue [If the break-in time was not successfully guessed, the impersonator does not continue.]

4⟩ `Return` SK_j to F and continue by running $F(\mathsf{forge}, SK_j)$. [Here $j = b_{\text{guess}}$.] During this execution, answer any hash-oracle queries of F [F might still make hash-oracle queries although it will no longer make sign-oracle queries] exactly as above.

5⟩ F now outputs its forgery $(M, \langle j, (Y, Z')\rangle)$ where $j < b_{\text{guess}}$. Execute $H_{\text{sim}}(j, Y, M)$ [Make sure that the simulated hash has an output value defined for this query.]

6⟩ `If` $(j, Y, M) \neq (j', Y', M')$ `then abort` [The values j', Y', M' were set by H_{sim} based on i_{guess}. If we guessed i_{guess} wrong then abort.] `Else return` Z' [This is the reply of the impersonator to the verifier's challenge received earlier during the execution of H_{sim}.]

Figure 5: Impersonation algorithm that uses a given forger F as a subroutine to impersonate the prover in the forward-secure ID scheme $\mathsf{FID}[k, l, T]$.

`Algorithm` $H_{\mathrm{sim}}(j, Y, M)$

1⟩ $hcount \leftarrow hcount + 1$

2⟩ `If` $hcount = i_{\mathrm{guess}}$

3⟩ `Then`

- $Y' \leftarrow Y$; $j' \leftarrow j$; $M' \leftarrow M$ [Save the call values as global variables to which the main impersonation routine will refer.]
- `If` $H(j, Y, M)$ is defined `then abort`
- `Else return` Y', j' as the output of the impersonator in this phase, and end this phase of the impersonator. [The impersonator chooses j' as the period at which it will impersonate the prover, and, playing the role of the prover, lets Y' be the first message it sends to the verifier in the protocol. It outputs these, and will now wait to receive a challenge vector from the verifier.]
- Receive challenge $c_1 \ldots c_l$ from the verifier and set $H(j, Y, M) = c_1 \ldots c_l$. [At this point the impersonation algorithm enters the next phase and must now seek a reply Z' to this challenge that will pass the period j' verification procedure with first message Y'.]

4⟩ `Else` [Here $hcount \neq i_{\mathrm{guess}}$ so we simply pick a random value as the answer to the hash oracle query, unless the answer is already defined.]

- `If` $H(j, Y, M)$ is not defined `then` let $c_1 \ldots c_l \stackrel{R}{\leftarrow} \{0,1\}^l$ and define $H(j, Y, M) = c_1 \ldots c_l$ `End If`

5⟩ `Return` $H(j, Y, M)$ [This is returned to the calling impersonation routine as the answer to the hash oracle query.]

`Algorithm` $Sgn_{\mathrm{sim}}(j, M)$ [Use zero-knowledge simulation technique for honest verifier to return a valid signature.]

1⟩ $hcount \leftarrow hcount + 1$

2⟩ Let $Z \stackrel{R}{\leftarrow} \mathsf{Z}_N^*$ and let $c_1 \ldots c_l \stackrel{R}{\leftarrow} \{0,1\}^l$. Define

$$Y \;=\; \frac{Z^{2^{T+1-j}}}{\prod_{i=1}^{l} U_i^{c_i}} \bmod N \;.$$

3⟩ `If` $H(j, Y, M)$ is defined `then abort`

4⟩ `Else`

- Define $H(j, Y, M) = c_1 \ldots c_l$
- `Return` $\langle j, (Y, Z)\rangle$ [This is returned to the calling impersonation routine as the answer to the sign oracle query.]

Figure 6: Hash-oracle simulator subroutine H_{sim} and sign-oracle simulator subroutine Sgn_{sim} called by the impersonator algorithm of Figure 5.

make the forger believe this, the impersonator will simulate the hash-oracle and will try to make sure that the value of H at j', Y', M' is set to the challenge the impersonator receives from the verifier.

This raises several difficulties. The forger makes hash-oracle queries at will, at any time in its execution. The impersonator has no way of identifying the one that corresponds to the forgery out of the sequence of q_{hash} queries made. So it will choose a value $i_{\text{guess}} \in \{1, \ldots, q_{\text{hash}}\}$ at random and hope that the i_{guess}-th hash query of the experiment (of executing F) is the one corresponding to the forgery. If it guesses wrong, it fails, but it has a $1/q_{\text{hash}}$ chance of guessing right.

This raises another difficulty. At the time F makes its i_{guess}-th query the impersonation algorithm must be in its second (impersonation) phase so that it can output Y', j' and receive a challenge from the verifier. To do this, it must already have output its request b for the period of which it wants the secret key SK_b. But at this time it may have no idea at what period F wants to break-in. It cannot wait until F tells it, because the hash oracle query needs answering and the impersonator needs to talk to the verifier for this. So it has no choice but to make its choice of b at random, right in the beginning, and hope this choice was right.

Assuming both random guesses are right, the execution of F will continue as planned subject to a final issue. In running F our impersonator must reply to hash-oracle queries and sign-oracle queries made by F. It does this via the subroutines of Figure 6. They build up the hash function H by defining its values on points as they are queried. The simulation of sign-oracle queries is done by using the idea in the zero-knowledge simulation of the honest verifier, taking advantage of the fact that the impersonator can first pick the hash function response and then define the point at which the hash function returns this value. This works as long as the hash function is not already defined at the point at which the sign-oracle simulator would like to define it. If the latter happens the routine fails, and the analysis must account for the probability of this happening.

The simulation of the hash-oracle is straightforward except that the routine is responsible for dealing appropriately with the "special" query, namely the i_{guess}-th one, by going to the verifier for the response. This routine could also fail if the simulated hash function is already defined at the point at which we would like to set the response to the verifier's challenge, and we will have to account for that too.

Analysis. We now analyze algorithm I to establish Equation (5). Step 5 of the impersonation routine in Figure 5 makes sure that the hash-oracle query corresponding to the forgery is made of the hash-oracle simulator at least once. (We would expect that it has been made prior to this, since otherwise the forger, not even knowing the challenge, has very little hope of success. But it is convenient to be sure that the hash-oracle simulator has processed the crucial query, so we make the call after the forgery anyway.) This means that i_{guess} has probability at least $1/q_{\text{hash}}$ of equalling the index (in the ordered list of queries made of the hash-oracle) that represents the first time this query was made. (We want the first time because otherwise H_{sim} will be unable to define the value and will abort.) The choice of b_{guess} made by the first phase of the impersonation algorithm certainly has chance at least $1/(T+1)$ of equalling the break-in time request of F. Both choices are made independently of others and the "view" of F is correct in the simulation subject to this execution not being aborted. Thus the success probability of I is at least

$$\frac{\mathbf{Succ}^{\text{fwsig}}(\mathsf{FSIG}[k,l,T],F) - p_a}{q_{\text{hash}} \cdot (T+1)},$$

where p_a is the probability of abortion arising from Sgn_{sim}. (That is the only source of abortion not accounted for. Abortion via H_{sim} was accounted for above by looking at the probability of guessing the first time the crucial query was made.) So now we claim that p_a is at most $(4q_{\text{hash}} \cdot q_{\text{sig}})/|Z_N^*|$.

We then (somewhat loosely) approximate $|Z_N^*|$ by 2^k to obtain Equation (5). It remains to justify the claim about the bound on p_a. The chance of $Sgn_{\rm sim}$ aborting on the i-th call to it ($i = 1, \ldots, q_{\rm sig}$) is at most

$$\frac{q_{\rm hash} - q_{\rm sig} - 1 + i}{|Z_N^*|/4} \, .$$

This is because in the worst case all the direct hash queries of F were made before the indirect (arising from the sign routine) ones, and we look at the number of hash-oracle queries made divided by the size of the domain over which Y is chosen. The random choice of Z makes Y a random quadratic residue since N is a Blum-Williams integer, so the denominator above is the number of quadratic residues in Z_N^*. Now for the total abortion probability we must sum the above over $i = 1, \ldots, q_{\rm sig}$, and this sum is at most

$$\frac{4(q_{\rm hash} - 1) \cdot q_{\rm sig}}{|Z_N^*|} \, .$$

Acknowledgments

We thank the Crypto 98 program committee and Michel Abdalla for their comments. We thank Victor Shoup for helpful discussions.

References

[1] R. Anderson, Invited lecture, *Fourth Annual Conference on Computer and Communications Security*, ACM, 1997.

[2] M. Bellare and S. Miner, "A forward-secure digital signature scheme," Preliminary version of this paper, *Advances in Cryptology – Crypto 99 Proceedings*, Lec. Notes in Comp. Sci. Vol. ??, M. Wiener ed., Springer-Verlag, 1999.

[3] M. Bellare and P. Rogaway, "Random oracles are practical: a paradigm for designing efficient protocols," *Proceedings of the First Annual Conference on Computer and Communications Security*, ACM, 1993.

[4] M. Bellare and P. Rogaway, "The exact security of digital signatures: How to sign with RSA and Rabin," *Advances in Cryptology – Eurocrypt 96 Proceedings*, Lec. Notes in Comp. Sci. Vol. 1070, U. Maurer ed., Springer-Verlag, 1996.

[5] G. R. Blakley, "Safeguarding cryptographic keys." *Proceedings of AFIPS 1979 National Computer Conference*, AFIPS, 1979.

[6] L. Blum, M. Blum and M. Shub, "A simple unpredictable pseudo-random number generator," *SIAM Journal on Computing* Vol. 15, No. 2, 364-383, May 1986.

[7] Y. Desmedt and Y. Frankel, "Threshold cryptosystems." *Advances in Cryptology – Crypto 89 Proceedings*, Lec. Notes in Comp. Sci. Vol. 435, G. Brassard ed., Springer-Verlag, 1989.

[8] W. Diffie, P. van Oorschot and M. Wiener, "Authentication and authenticated key exchanges," *Designs, Codes and Cryptography*, 2, 107–125 (1992).

[9] U. Feige, A. Fiat, and A. Shamir, "Zero-knowledge proofs of identity," *J. of Cryptology*, 1(1988), 77-94.

[10] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," *Advances in Cryptology – Crypto* 86 *Proceedings*, Lec. Notes in Comp. Sci. Vol. 263, A. Odlyzko ed., Springer-Verlag, 1986.

[11] S. Goldwasser, S. Micali, and C. Rackoff, "The Knowledge Complexity of Interactive Proof Systems", *SIAM J. on Computing*, Vol. 18, No. 1, 1989, pp. 186–208.

[12] S. Goldwasser, S. Micali and R. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal of Computing*, Vol. 17, No. 2, pp. 281–308, April 1988.

[13] C. Günther, "An identity-based key-exchange protocol," *Advances in Cryptology – Eurocrypt* 89 *Proceedings*, Lec. Notes in Comp. Sci. Vol. 434, J-J. Quisquater, J. Vandewille ed., Springer-Verlag, 1989.

[14] S. Haber and W. Stornetta, "How to Time-Stamp a Digital Document," *Advances in Cryptology – Crypto* 90 *Proceedings*, Lec. Notes in Comp. Sci. Vol. 537, A. J. Menezes and S. Vanstone ed., Springer-Verlag, 1990.

[15] A Herzberg, M. Jakobsson, S .Jarecki, H Krawczyk and M. Yung, "Proactive public key and signature schemes," *Proceedings of the Fourth Annual Conference on Computer and Communications Security*, ACM, 1997.

[16] K. Ohta and T. Okamoto. "On concrete security treatment of signatures derived from identification," *Advances in Cryptology – Crypto* 98 *Proceedings*, Lec. Notes in Comp. Sci. Vol. 1462, H. Krawczyk ed., Springer-Verlag, 1998.

[17] H. Ong and C. Schnorr, "Fast signature generation with a Fiat-Shamir like scheme," *Advances in Cryptology – Eurocrypt* 90 *Proceedings*, Lec. Notes in Comp. Sci. Vol. 473, I. Damgård ed., Springer-Verlag, 1990.

[18] D. Pointcheval and J. Stern, "Security proofs for signature schemes," *Advances in Cryptology – Eurocrypt* 96 *Proceedings*, Lec. Notes in Comp. Sci. Vol. 1070, U. Maurer ed., Springer-Verlag, 1996.

[19] J. Rompel, "One-Way Functions are Necessary and Sufficient for Secure Signatures," *Proceedings of the* 22nd *Annual Symposium on Theory of Computing*, ACM, 1990.

[20] A. Shamir, "How to share a secret," *Communications of the ACM*, 22(1979), 612-613.

[21] V. Shoup, "On the security of a practical identification scheme," *Advances in Cryptology – Eurocrypt* 96 *Proceedings*, Lec. Notes in Comp. Sci. Vol. 1070, U. Maurer ed., Springer-Verlag, 1996.

[22] H. Williams, "A Modification of the RSA Public-key Encryption Procedure," *IEEE Transactions on Information Theory*, Vol. IT-26, No. 6, 1980, pp. 726–729.