The extended abstract of this work appears *Advances in Cryptology — Asiacrypt 2000*, Tatsuaki Okamoto, editor, Lecture Notes in Computer Science vol. 1976, Springer-Verlag, 2000. © IACR

A New Forward-Secure Digital Signature Scheme

MICHEL ABDALLA* LEONID REYZIN†

November 17, 2000

Abstract

We improve the Bellare-Miner (Crypto '99) construction of signature schemes with forward security in the random oracle model. Our scheme has significantly shorter keys and is, therefore, more practical. By using a direct proof technique not used for forward-secure schemes before, we are able to provide better security bounds for the original construction as well as for our scheme.

Bellare and Miner also presented a method for constructing such schemes without the use of the random oracle. We conclude by proposing an improvement to their method and an additional, new method for accomplishing this.

Keywords: forward security, digital signatures, proven security, concrete security.

*Dept. of Computer Science & Engineering, University of California at San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. E-Mail: `mabdalla@cs.ucsd.edu`. URL: `http://www-cse.ucsd.edu/users/mabdalla`.
Supported by CAPES under Grant BEX3019/95-2.

† MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139, USA. E-Mail: `reyzin@mit.edu`. URL: `http://theory.lcs.mit.edu/~reyzin`.
Supported by the National Science Foundation Graduate Research Fellowship and a grant from the NTT corporation.

Contents

1 Introduction

1.1 The problem

Many cryptographic techniques today, whether only available in the literature or actually used in practice, are believed to be quite secure. Several, in fact, can be proven secure (with appropriate definitions) under very reasonable assumptions. In a vast majority of solutions, however, security guarantees last only as long as secrets remain unrevealed. If a secret is revealed (either accidentally or via an attack), security is often compromised not only for subsequent uses of the secret, but also for prior ones. For example, if a secret signing key becomes known to an adversary, one cannot trust any signature produced with that key, regardless of when; if a secret decryption key becomes known to an adversary, then any encrypted message, even if sent long before, is not guaranteed to remain private.

To address this problem, several different approaches have been suggested. Many attempt to lower the chance of exposure of secrets by distributing them across several systems, usually via secret sharing. As pointed out in [2], this method is usually quite costly, and may, in fact, be too expensive to be implemented by a typical individual user. Moreover, since each of the systems may be susceptible to the same attack, the actual risk may not decrease.

A complementary approach is to reduce the potential damage in case secrets are exposed. In what is often called *forward security*, the main idea is to ensure that secrets are used only for short time periods, and that compromise of a secret does not affect anything based on secrets from prior time periods. One of the challenges in designing such a system is to be able to change secret information without the inconvenience of changing public information, such as the public key.

This approach has been known in the context of key agreement as *forward secrecy* [13, 7]. In the context of digital signatures, it was first proposed, together with a few simple solutions, by Anderson in [1]. Bellare and Miner formalized Anderson's approach and provided more solutions in [2]

The specific problem addressed in this paper is that of designing a forward-secure signature scheme.

1.2 Forward-Secure Signature Schemes

Informally, a *key-evolving signature scheme* is one whose operation is divided into time periods, with a different secret key for each time period. Each secret key is used to sign messages only during a particular time-period, and to compute a new secret key at the end of that time period. It is then erased. As in ordinary signature schemes, however, there is only one public key, which remains the same through all the time periods. The verification algorithm checks not only that a signature is valid, but also that it was generated during a specific time period.

Such a scheme is *forward-secure* if it is infeasible for an adaptive chosen-message adversary to forge signatures for past time periods, even if it discovers the secret key for the current time period. Note that, in particular, this implies that past secret keys cannot be recovered from the current one. In a forward-secure signature scheme, even if the current secret key is compromised, signatures from past time periods can still be trusted.

Anderson [1] proposed a construction of forward-secure signature schemes in which the size of secret key (but not the public key) grows linearly with the number of time periods. The first forward-secure signature schemes in which key sizes do not grow linearly were proposed by Bellare and Miner in [2]. Their most efficient scheme, forward-secure in the random oracle model of [3] (assuming factoring is hard), uses ideas from the Fiat-Shamir [9] and Ong-Schnorr [15] identification and signature schemes.

As mentioned in [2], although still practical, their scheme requires very large keys, mainly because the original Fiat-Shamir scheme required very large keys (in fact, the forward-secure scheme of [2] does not add much to the already large key).

1.3 Our Contributions

MAIN RESULT. We propose a new forward-secure digital signature scheme, with much shorter keys than those in the scheme of [2]. In fact, our keys are comparable in size to those used in similar ordinary signature schemes.

Similarly to the scheme of [2], our scheme is based on signature schemes that are derived from three-round identification protocols. Specifically, the scheme is based on a generalized version of Micali's signature scheme [16], which is in many ways similar to the schemes of Ong-Schnorr [15], Guillou-Quisquater [12] and Ohta-Okamoto [18]. It is quite simple and efficient, although the computational efficiency of some components is less than that of the scheme of [2]. Our scheme can also be proven forward secure in the random oracle model, assuming factoring is hard.

OTHER CONTRIBUTIONS. While [2] use reduction to identification schemes to prove security, we use a direct proof technique. This enables us to provide a tighter exact security analysis for our scheme than the indirect technique of [2]. In fact, our technique can also be applied to the scheme of [2] to obtain a tighter security analysis for that scheme (which we present in Section 3.5).

We also present methods of achieving forward security in signature schemes without relying on random oracles. In general, they are less efficient than our main construction, and are not practical. However, they are still of interest, and can be viewed as an improvement on the tree-based construction of [2].

1.4 Outline

We start by giving in Section 2 precise definitions of key-evolving signature schemes and their forward-security. We then proceed in Section 3 with the description of our main scheme, its security analysis, and its comparison to the scheme of Bellare and Miner. We conclude in Section 4 by discussing alternate methods for achieving forward security in signature schemes, which do not rely on random oracles.

2 Definitions

In this section, we first describe how a forward secure digital signature scheme operates and define its formal notion of security, both inside and outside the random oracle model. All definitions provided here are based on those given in [2], which in turn are based on those given in [11] and [4]. Since our scheme is based on the hardness of factoring Blum integers, we also present a formal definition for the security of this problem.

2.1 Forward-secure digital signature schemes

A forward-secure digital signature scheme is, first of all, a key-evolving digital signature scheme. A key-evolving signature scheme is very similar to a standard one. Like a standard signature scheme, it contains a key generation algorithm, a signing algorithm, and a verification algorithm. The public key is left unchanged throughout the lifetime of the scheme, making the verification algorithm very similar to that of a standard signature scheme. Unlike a standard signature scheme, a key-evolving signature scheme has its operation divided into time periods, each of which uses a different (but related) secret key to sign a message. The way these keys are updated is given by a public update algorithm, which computes the secret key for the new time period based on that for the previous one. The forward security comes, in part, from the fact that this update function is one-way and, given the secret key for the current period, it is hard to compute any of the previously used secret keys. It is important, of course, for the signer to delete the old secret key as soon as the new one is generated, since otherwise an adversary breaking the system could easily get hold of these undeleted keys and forge messages for time periods preceding that of the break-in. Let us now define more formally what a key-evolving digital signature scheme is and then define what it means for it to be forward-secure.

Definition 2.1 [Key-evolving signature scheme] A *key-evolving digital signature scheme* is a quadruple of algorithms, $\mathsf{FSIG} = (\mathsf{FSIG.key}, \mathsf{FSIG.update}, \mathsf{FSIG.sign}, \mathsf{FSIG.vf})$, where:

- $\mathsf{FSIG.key}$, the *key generation* algorithm, is a probabilistic algorithm which takes as input a security parameter $k \in \mathsf{N}$ (given in unary as 1^k) and the total number of periods T and returns a pair (SK_0, PK), the initial secret key and the public key;
- $\mathsf{FSIG.sign}$, the (possibly probabilistic) *signing* algorithm, takes as input the secret key SK_j for the current time period and the message M to be signed and returns a pair $\langle j, sign\rangle$, the signature of M for time period j;
- $\mathsf{FSIG.update}$, the (possibly probabilistic) *secret key update* algorithm, takes as input the secret key for the current period SK_j and returns the new secret key SK_{j+1} for the next period.
- $\mathsf{FSIG.vf}$, the (deterministic) *verification* algorithm, takes as input the public key PK, a message M, and a candidate signature $\langle j, sign\rangle$, and returns 1 if $\langle j, sign\rangle$ is a *valid* signature of M or 0, otherwise. It is required that $\mathsf{FSIG.vf}_{PK}(M, \mathsf{FSIG.sign}_{SK_j}(M)) = 1$ for every message M and time period j.

We also assume that the secret key SK_j for time period $j \leq T$ always contains both the value j itself and the value T of the total number of periods. Finally, we adopt the convention that SK_{T+1} is the empty string and that $\mathsf{FSIG.update}_{SK_T}$ returns SK_{T+1}. ▌

When we work in the random oracle model, all the above-mentioned algorithms would additionally have oracle access to a public hash function H, which is assumed to be random in the security analysis.

Security in the standard model. Informally, we want it to be computationally infeasible for any adversary to forge a signature with respect to any of the previously used secret keys even in the event of exposure of the current secret key. Of course, since the update algorithm is public, nothing can be done with respect to future secret keys, except for revoking the public key and considering invalid any signature for the time period of the break-in and thereafter. To define this notion of security formally, we again use the security model introduced by Bellare and Miner [2], which extends that of Goldwasser, Micali and Rivest [11] to take into account the ability of the adversary to obtain a key by means of a break-in.

In this new model, besides knowing the user's public key PK, the adversary also gets to know the total number of time periods and the current time period. The adversary runs in three phases. In the first phase, the chosen message attack phase (cma), the adversary has access to a signing oracle, which it can query to obtain signatures of messages of its choice with respect to the current secret key. At the end of each time period, the adversary can choose whether to stay in the same phase or switch to the break-in phase ($\mathsf{breakin}$). In the break-in phase, which models the possibility of a key exposure, we give the adversary the secret key SK_j for the specific time period j it decided to break in. In the last phase, the forgery phase (forge), the adversary outputs a pair signature-message, that is, a forgery. The adversary is considered to be successful if it forges a signature of some new message (that is, not previously queried to the signing oracle) for some time period prior to j.

In order to capture the notion of forward security of a key-evolving signature scheme $\mathsf{FSIG} = (\mathsf{FSIG.key}, \mathsf{FSIG.update}, \mathsf{FSIG.sign}, \mathsf{FSIG.vf})$ more formally, let F be an adversary for this scheme. To assess the success probability of F breaking the forward security of FSIG, consider the following experiment.

Experiment F-Forge(FSIG, F)
$\quad (PK, SK_0) \stackrel{R}{\leftarrow} \mathsf{FSIG.key}(k, \ldots, T)$
$\quad j \leftarrow 0$
$\quad$ `Repeat`
$\qquad j \leftarrow j+1\ ;\ SK_j \leftarrow \mathsf{FSIG.update}(SK_{j-1})\ ;\ d \leftarrow F^{\mathsf{FSIG.sign}_{SK_j}(\cdot)}(\mathsf{cma}, PK)$
$\quad$ `Until` $(d = \mathsf{breakin})$ or $(j = T)$
$\quad$ `If` $d \neq \mathsf{breakin}$ and $j = T$ `then` $j \leftarrow T+1$
$\quad (M, \langle b, sign \rangle) \leftarrow F(\mathsf{forge}, SK_j)$
$\quad$ `If` $\mathsf{FSIG.vf}_{PK}(M, \langle b, sign \rangle) = 1$ `and` $1 \leq b < j$
$\qquad$ `and` M was not queried of $\mathsf{FSIG.sign}_{SK_b}(\cdot)$ in period b
$\qquad$ `then return` 1 `else return` 0

It is understood above that the state of F is preserved between invocations and we can fix its coins. Also, if the adversary F does not break-in by the last period, we return to it the empty string as the secret key for the time period $T+1$.

Definition 2.2 [Forward-security in standard model] Let $\mathsf{FSIG} = (\mathsf{FSIG.key}, \mathsf{FSIG.update}, \mathsf{FSIG.sign}, \mathsf{FSIG.vf})$ be a key-evolving signature scheme and F an adversary as described above. Let $\mathbf{Succ}^{\mathrm{fwsig}}(\mathsf{FSIG}[k, \ldots, T], F)$ denote the probability that experiment F-Forge$(\mathsf{FSIG}[k, \ldots, T], F)$ returns 1. Then the insecurity of FSIG is the function

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

where the maximum here is taken over all adversaries F making a total of at most q_{sig} queries to the signing oracles across all the stages and for which the above experiment runs in time at most t. ▮

It is understood that the running time t also accounts for the time to answer oracle queries.

The insecurity function above follows the concrete security paradigm and gives us a measure of how secure or insecure the scheme really is. Therefore, we want its value to be as small as possible. Our goal in a security proof will be to find an upper bound for it.

Security in the RO model. In case we are working in the RO model, we have to modify the experiment F-Forge to account for the queries made to the random oracle H and add to Definition 2.2 a new parameter, q_{hash}, the total number of queries to H-oracle in the experiment. Thus, in order to define what it means for a key-evolving signature scheme to be secure in the RO model, let us first define the following experiment.

Experiment F-Forge-RO(FSIG, F)
$\quad$ Select $H\colon \{0,1\}^* \to \{0,1\}^l$ at random
$\quad (PK, SK_0) \stackrel{R}{\leftarrow} \mathsf{FSIG.key}^H(k, \ldots, T)$

$j \leftarrow 0$
`Repeat`
$\quad j \leftarrow j+1$; $SK_j \leftarrow \mathsf{FSIG.update}^H(SK_{j-1})$; $d \leftarrow F^{H,\mathsf{FSIG.sign}^H_{SK_j}(\cdot)}(\mathsf{cma}, PK)$
`Until` $(d = \mathsf{breakin})$ or $(j = T)$
`If` $d \neq \mathsf{breakin}$ and $j = T$ `then` $j \leftarrow T+1$
$(M, \langle b, sign \rangle) \leftarrow F^H(\mathsf{forge}, SK_j)$
`If` $\mathsf{FSIG.vf}^H_{PK}(M, \langle b, sign \rangle) = 1$ `and` $1 \leq b < j$
$\quad$ `and` M was not queried of $\mathsf{FSIG.sign}^H_{SK_b}(\cdot)$ in period b
$\quad$ `then return` 1 `else return` 0

Definition 2.3 [Forward-security in RO model] Let $\mathsf{FSIG} = (\mathsf{FSIG.key}, \mathsf{FSIG.update}, \mathsf{FSIG.sign}, \mathsf{FSIG.vf})$ be a key-evolving signature scheme, H be a random oracle and F an adversary as described above. We let $\mathbf{Succ}^{\text{fwsig}}(\mathsf{FSIG}[k, \ldots, T], F)$ denote the probability that the experiment F-Forge-RO$(\mathsf{FSIG}[k, \ldots, T], F)$ returns 1. Then the insecurity of FSIG is the function

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

where the maximum here is taken over all adversaries F making a total of at most q_{sig} queries to the signing oracles across all the stages and for which the running time of the above experiment is at most t and at most q_{hash} queries are made to the random oracle H. ▮

2.2 Factoring

Let A be an adversary for the problem of factoring Blum integers. That is, A gets as input an integer N that is the product of two primes, each congruent to 3 modulo 4, and tries to compute these prime factors. We define the following experiment using notation from [2].

Experiment Factor(k, A)
Randomly choose two primes p and q, such that:
$\quad p \equiv q \equiv 3 \pmod 4$, $2^{k-1} \leq (p-1)(q-1)$, and $pq < 2^k$
$N \leftarrow pq$
$(p', q') \leftarrow A(N)$
`If` $p'q' = N$ `and` $p' \neq 1$ `and` $q' \neq 1$ `then return` 1 `else return` 0

Definition 2.4 [Factoring] Let A be an adversary for the problem of factoring Blum integers and let $\mathbf{Succ}^{\text{fac}}(A, k)$ denote the probability that experiment Factor(k, A) returns 1. The insecurity of factoring Blum integers is the function

$$\mathbf{InSec}^{\text{fac}}(k, t) \;=\; \max_A \{ \mathbf{Succ}^{\text{fac}}(A, k) \} \;,$$

where the maximum here is taken over all adversaries A for which the above experiment runs in time at most t. ▮

3 Our scheme

We start by explaining some number theory that provides intuition for our construction. We then present a slight variation of a signature scheme due to Micali [16]. The scheme has similarities to the schemes of Ong-Schnorr [15], Guillou-Quisquater [12] and Ohta-Okamoto [18] and, like they, is based on the idea of Fiat and Shamir [9] for converting identification schemes into signature schemes.

We then modify the signature scheme to make it forward-secure, and prove its security.

The schemes in this section are in the random oracle model. We will call the oracle $H : \{0,1\}^* \to \{0,1\}^l$.

3.1 Number Theory

Let k and l be two security parameters. Let $p_1 \equiv p_2 \equiv 3 \pmod 4$ be two primes of approximately equal size and $N = p_1 p_2$ be a k-bit integer (such N is called a *Blum* integer).To simplify further computations, we will assume not only that $N > 2^{k-1}$, but also that $|Z_N^*| = N - p_1 - p_2 + 1 \geq 2^{k-1}$. Let Q denote the set of non-zero quadratic residues modulo N. Note that $|Q| \geq 2^{k-3}$. Note also that for $x \in Q$, exactly one of its four square roots is also in Q (this follows from the fact that -1 is a non-square modulo p_1 and p_2 and the Chinese remainder theorem). Thus, squaring is a permutation over Q. From now on, when we speak of "the square root of x," we mean the single square root in Q; by $x^{2^{-k}}$ we will denote the single $y \in Q$ such that $x = y^{2^k}$.

Let $U \in Q$. Following [11], define $F_0(Z) = Z^2 \bmod N$, $F_1(Z) = UZ^2 \bmod N$, and, for an l-bit binary string $\sigma = b_1 \dots b_l$, define $F_\sigma : Q \to Q$ as $F_\sigma(Z) = F_{b_l}(\dots(F_{b_2}(F_{b_1}(Z)))\dots) = Z^{2^l} U^\sigma \bmod N$ (note that U^σ is a slight abuse of notation, because σ is a binary string, rather than an integer; what is really meant here is U raised to the power of the integer represented in binary by σ). Because squaring is a permutation over Q and $U \in Q$, F_σ is a permutation over Q.

Note that $F_\sigma(Z)$ can be efficiently computed by anybody who knows N and U. Also, if one knows p_1 and p_2, one can efficiently compute $Z = F_\sigma^{-1}(Y)$ for a given Y (as shown by Goldreich in [10]) by computing $S = 1/U^{2^{-l}} \bmod N$ and then letting $Z = Y^{2^{-l}} S^\sigma \bmod N$ (these calculations can be done modulo p_1 and p_2 separately, and the results combined using the Chinese remainder theorem). However, if one does not know the square root of U, then F_σ^{-1} is hard to compute, as shown in the Lemma below (due to [11]).

Lemma 3.1 Given $Y \in Q$, two different strings σ and τ of equal length, $Z_1 = F_\sigma^{-1}(Y)$ and $Z_2 = F_\tau^{-1}(Y)$, one can compute $V \in Q$ such that $V^2 \equiv U \pmod N$.

Proof: The proof is by induction on the length of the strings σ and τ.

If $|\sigma| = |\tau| = 1$, then assume, without loss of generality, that $\sigma = 0$ and $\tau = 1$. Then $F_0(Z_1) = F_1(Z_2) = Y$, i.e., $Z_1^2 \equiv UZ_2^2 \pmod N$, so we can set $V = Z_1/Z_2 \bmod N$.

For the inductive case, let σ and τ be two strings of length $m+1$. Let σ' and τ' be their m-bit prefixes, respectively. If $F_{\sigma'}(Z_1) = F_{\tau'}(Z_2)$, we are done by the inductive hypothesis. Otherwise, the last bit of σ must be different from the last bit of τ, so, without loss of generality, assume the last bit of σ is 0 and the last bit of τ is 1. Then $F_0(F_{\sigma'}(Z_1)) = F_1(F_{\tau'}(Z_2))$, and the same proof as for the base case works here. ▌

We will now provide a geometric interpretation of the discussion above. Consider a complete binary tree of depth l where each node stores a value in Q. The root (at the top of the tree) stores Y. The values at the children of a node that stores A are $F_0^{-1}(A)$ at the left child and $F_1^{-1}(A)$ at the right child. Then computing $F_\sigma^{-1}(Y)$ means finding the value at the leaf for which the path from the root is given by σ (where right-to-left in σ corresponds to top-to-bottom in the tree).

It is clearly easy to compute the values "up" the tree from a given node. What the lemma says is that it is hard to compute the values "down" the tree without the ability to take square roots: in fact, if one knows two paths from the bottom of the tree, then one can get the square root of U by looking at the children of the point where the two paths join together.

Finally, note that the value R stored at the bottom-left leaf of tree is $F_{00\dots0}^{-1}(Y) = Y^{2^{-l}}$, so if one knows $S = 1/U^{2^{-l}}$ and R, then one can compute the value at any leaf (given by σ) by computing $RS^\sigma \bmod n$.

3.2 The 2^l-th root signature scheme

The discussion above suggests the following signature scheme, which is similar to the schemes of [15] and [16] (an interactive three-round identification scheme can be designed similarly).

The signer generates a modulus N, picks a random $S \in Q$ to keep as its secret key, computes $U = 1/S^{2^l}$ and outputs (N, U) as its public key.

To sign a message M, it first generates a random $R \in Q$ and computes $Y = R^{2^l}$. Note that this gives it the ability to find any leaf of the binary tree described above, rooted at Y. It therefore computes $\sigma = H(Y, M)$ and $Z = F_\sigma^{-1}(Y) = RS^\sigma \bmod N$ which it outputs as the signature.

The verifier checks that $Z \not\equiv 0 \pmod N$ and computes $Y' = F_\sigma(Z) = Z^{2^l} U^\sigma \pmod N$. It then verifies that $\sigma = H(Y', M)$.

We will not prove the security of this scheme here. The intuition, however, is the following: the verifier believes the signature because the signer was able to go down a random (given by H) path in the tree rooted at Y. Because the ability to go down two different paths implies the knowledge of the square root of U, the ability to go down a random path out of 2^l probably also implies that knowledge.

One point worth mentioning is that the verifier does not know if $U, Z \in Q$. All it knows is that $U, Z \not\equiv 0 \pmod N$, so either $U, Z \in Z_N^*$ or else one of the gcd's (U, N), (Z, N) gives a factorization of N. We therefore need the following reformulation of Lemma 3.1.

Lemma 3.2 Given $Z_1, Z_2, U \in Z_N^*$ and two different strings σ and τ of equal length such that $Z_1^{2^l} U^\sigma \equiv Z_2^{2^l} U^\tau \pmod N$, one can compute $V \in Z_N^*$ such that $V^2 \equiv U \pmod N$.

Proof: The proof is the same as for Lemma 3.1. ▮

In fact, now that we have this lemma, S and R picked by the signer need not be in Q: they can come from Z_N^*.

3.3 The Forward-Secure Signature Scheme

Note that the security of the above scheme hinges on the value S and the number l of squaring operations that separates it from U. It is S that allows the signer to go from the leftmost leaf of the tree to any leaf and it is l that determines the maximum depth of the tree.

Thus, a reasonable way of making the scheme forward-secure is to start out with a deep tree, and to use smaller and smaller depths for subsequent time periods. Then new values of S can be obtained from old values of S simply by squaring. Old values of S cannot be recovered from new ones.

While making the tree deeper, however, there is no need to make it any wider. The width of the tree is only used to ensure that σ is sufficiently random, so the adversary cannot guess what σ will be and thus forge a signature. Therefore, the tree will remain complete to a certain sufficient depth, and from that point, each node will only have the left child (given by F_0^{-1}). The length of σ will remain the same (l). This will make the scheme more efficient.

Now there is a question of how much up the tree we should go with each time period (that is, by how many squarings the current value of S should be separated from the previous value S'). Note that, in order to compute a signature with respect to S', one only needs S'^σ, not S' itself. Thus, if $S \equiv S'^{2^x} \pmod N$ and the last x bits of σ are 0, then S will allow one to compute the signature. Therefore, we should separate S from S' by $|\sigma|$ squarings, so that a forgery is possible for exactly one value of σ, as before. A smaller separation makes no sense without the corresponding reduction in the length of σ and, therefore, the width of tree.

```
algorithm FSIG.key(k, T)
begin
  Generate random primes p1, p2 such that:
    p1 ≡ p2 ≡ 3 (mod 4)
    2^(k-1) ≤ (p1 - 1)(p2 - 1)
    p1 p2 < 2^k
  N ← p1 p2
  S0 ←R Z*_N
  U ← 1/S0^(2^(l(T+1))) mod N
  SK ← (N, T, 0, S0)
  PK ← (N, U, T)
  return (S, U)
end
```

```
algorithm FSIG.update(SK)
begin
  parse SK as (N, T, j, S_j)
  if j = T then
    SK ← ε
  else
    SK ← (N, T, j + 1, S_j^(2^l) mod N)
  return SK
end
```

```
algorithm FSIG.sign^H(M, SK)
begin
  parse SK as (N, T, j, S_j)
  R ←R Z*_N
  Y ← R^(2^(l(T+1-j))) mod N
  σ ← H(j, Y, M)
  Z ← R S_j^σ mod N
  return (j, (Z, σ))
end
```

```
algorithm FSIG.vf^H(M, PK, sign)
begin
  parse PK as (N, U, T)
  parse sign as (j, (Z, σ))
  if Z ≡ 0 (mod N)
    return 0
  else
    Y' ← Z^(2^(l(T+1-j))) U^σ mod N
    if σ = H(j, Y', M) then
      return 1
    else
      return 0
end
```

Figure 1: *Our forward-secure digital signature scheme*

Having given the intuition, we refer the reader to Figure 1 for the complete description of our forward-secure scheme.

3.4 Security Analysis

We state the following theorem that will allow us to upper-bound the insecurity function for this signature scheme. Its proof combines ideas from [19], [2] and [17]. The proof technique used here can also be used to improve the bound on the insecurity function of the forward-secure scheme of [2] (see Section 3.5 for more details).

Theorem 3.3 If there exists a forger F for $\mathsf{FSIG}[k,l,T]$ that runs in time at most t, asking at most q_{hash} hash queries and q_{sig} signing queries, such that $\mathbf{Succ}^{\text{fwsig}}(\mathsf{FSIG}[k,l,T],F) \geq \varepsilon$, then there exists an algorithm A that factors Blum integers generated by $\mathsf{FSIG.key}(l,T)$ in expected time at most t' with probability at least ε', where

$$t' = 2t + O(k^2 lT + k^3)$$

$$\varepsilon' = \frac{\left(\varepsilon - 2^{3-k} q_{\text{sig}}(q_{\text{hash}}+1)\right)^2}{2T^2(q_{\text{hash}}+1)} - \frac{\varepsilon - 2^{3-k} q_{\text{sig}}(q_{\text{hash}}+1)}{2^{l+1}T}.$$

Proof Idea. To factor its input N, A will select a random $x \in Z_N^*$, compute $v = x^2 \bmod N$, and attempt to use the adversary to find a square root y of v. Because v has four square roots and x is random, with probability $1/2$ we have that $x \not\equiv \pm y \pmod N$ and, hence A will be able to find a factor of N by computing the gcd of $x-y$ and N.

So, the task of A is to find a square root of v without using x. Note that A gets to provide the public key for F and to answer its signing and hashing queries. The idea, then, is to base to the public key U on v and run F once to get a signature $(b,(Z,\sigma))$. Note that F had to ask a hash query on (b,Y,M) where $Y = Z^{2^{k(T+1-b)}}U^{\sigma}$—otherwise, the probability of its correctly guessing σ is at most 2^{-l}. Then, run F the second time with the same random tape, giving the same answers to all the oracle queries before the query (b,Y,M). For (b,Y,M) give a new answer τ. Then, if F again forges a signature $(b,(Z',\tau))$ using Y and M, we will have a condition similar to that of Lemma 3.2, and will be able to compute a square root of v. Please refer to Appendix A for the actual proof.

Theorem 3.4 Let $\mathsf{FSIG}[k,l,T]$ represent our key evolving signature scheme with modulus size k, challenge length l, and number of time periods T. Then for any t, q_{sig}, and q_{hash},

$$\begin{aligned}&\mathbf{InSec}^{\text{fwsig}}(\mathsf{FSIG}[k,l,T];t,q_{\text{sig}},q_{\text{hash}}) \leq \\ &\qquad T\sqrt{2(q_{\text{hash}}+1)\mathbf{InSec}^{\text{fac}}(k,t')} + 2^{-l}T(q_{\text{hash}}+1) + 2^{3-k}q_{\text{sig}}(q_{\text{hash}}+1)\ ,\end{aligned}$$

where $t' = 2t + O(k^3 + k^2 lT)$.

Proof: The insecurity function is computed simply by solving for $(\varepsilon - 2^{3-k}q_{\text{sig}}(q_{\text{hash}}+1))/T$ the quadratic equation in Theorem 3.3 that expresses ε' in terms of ε to get

$$\begin{aligned}(\varepsilon - 2^{3-k}q_{\text{sig}}(q_{\text{hash}}+1))/T &= 2^{-l-1}(q_{\text{hash}}+1) + \sqrt{2^{-2l-2}(q_{\text{hash}}+1)^2 + 2\varepsilon'(q_{\text{hash}}+1)} \\ &\leq 2^{-l-1}(q_{\text{hash}}+1) + \sqrt{2^{-2l-2}(q_{\text{hash}}+1)^2} + \sqrt{2\varepsilon'(q_{\text{hash}}+1)} \\ &= 2^{-l}(q_{\text{hash}}+1) + \sqrt{2\varepsilon'(q_{\text{hash}}+1)},\end{aligned}$$

and then solving the resulting inequality for ε. ▮

3.5 Discussion

Note that, for any reasonable choices of q_{sig} and q_{hash}, the minimally secure value for the modulus size k (which should be greater than 512) makes the term $2^{3-k}q_{\text{sig}}(q_{\text{hash}}+1)$ negligible. The term $2^{-l}T(q_{\text{hash}}+1)$ allows one to find a value for l (the size of the hash values) that depends, mainly, on q_{hash} (which is the number of hash values an adversary is believed to be capable of computing). Finally, the term $T\sqrt{2(q_{\text{hash}}+1)\mathbf{InSec}^{\text{fac}}(k,t')}$ allows one to find the value for k that depends, mainly, on the assumed insecurity of factoring and on q_{hash} (because T, which is related to the efficiency of the scheme, is probably much less than q_{hash}).

Using our direct proof technique, the bound on the insecurity of the scheme of [2] can be improved by a factor of almost $\sqrt{Tq_{\text{hash}}}$ ([2] lose this factor by using an indirect proof, which first reduces the security of the signature scheme to the security of the corresponding identification scheme). The resulting bound is

$$T\sqrt{2l(q_{\text{hash}}+1)\mathbf{InSec}^{\text{fac}}(k,t')} + 2^{-l}T(q_{\text{hash}}+1) + 2^{3-k}q_{\text{sig}}(q_{\text{hash}}+1)\ ,$$

which is worse than that of our scheme by a factor of no more than $\sqrt{l}$. Thus, the two schemes have almost the same security for the same parameters $l, k, q_{\text{sig}}, q_{\text{hash}}$.

The size of both the public and the private keys in the scheme of [2] is about $k(l+1)$ bits, while the size of the keys is in our scheme is about $2k$ bits. So the keys in our scheme are about $(l+1)/2$ times shorter.

The efficiency of key generation and update algorithms is about the same for both schemes.

Signing for both scheme can be decomposed into two components: off-line (before the message is known) and on-line (once the message is available). The off-line component for time period j for the scheme of [2] takes time $T-j+1$ modular squarings, while for our scheme it takes l times more. The on-line component takes about $l/2$ multiplications for [2] and $3l/2$ for our scheme. However, because the on-line signing component in our scheme involves exponentiation of a fixed based, precomputation techniques are available. Specifically, if the signer, using a variation of the technique of Lim and Lee [14], precomputes 3 additional powers of S_j at the cost of increasing the secret key size by a factor of 2.5, the on-line component will take about $l/2$ multiplications—as long as in the [2] scheme. Precomputation of more values will reduce the on-line component of signing even further, at the expense of the secret key length and the efficiency of the update algorithm.

Finally, verification for time period j for the scheme of [2] takes about $T+1-j+l/2$ modular multiplications, while in our scheme about $l(T+1-j)+3l/2$ modular multiplications are needed. Again, precomputing powers of the public key may be used to reduce the $3l/2$ term, but this term is not very significant unless j is close to T.

Thus, our scheme has slightly better security, much shorter keys, and comparable efficiency for the on-line component of signing. The efficiency of the off-line component of signing and that of verifying is worse, however. Because each secret key needs to be separated by l squarings from the previous one (Section 3.3), we believe that the efficiency of off-line signing and verifying cannot be improved without a significant change in the design idea.

4 Schemes in the standard model

Both our scheme above and the Bellare-Miner's scheme were proven secure based on the hardness of factoring and on the assumption that the hash function H behaves like a random function. The main reason for this is that, when converting an identification scheme to a signature scheme (á la Fiat-Shamir [9]), the challenge produced by the hash function should be as random as that produced by an honest verifier, so as to maintain the security of this transformation.

One way of avoiding random oracles in the design of forward-secure signature schemes is to use the binary certification tree method suggested by Bellare and Miner [2]. It works as follows. Each node of the tree represents a pair of keys, a secret key and the related public key, used for an (ordinary) signature scheme. At the leaf level, each key is associated to a certain time period. Thus, the total number of leaves equals the total number of time periods. Each key at an internal node is used to certify the keys of its two children. The public key for the forward-secure scheme is the public key at the root of the tree. To sign a message in a certain time period, we use the secret key of the corresponding leaf and attach to the signature a certification chain based on the path from the root to that leaf so that the verifier can check the validity of the key itself. To maintain forward security, nodes are created dynamically. The secret key of an internal node is deleted as soon as it certifies the keys of its children. At any time, we only keep those keys on the path from the root to the leaf associated to the current time period, plus the right sibling of those nodes which are the left child of their parents. Consequently, as Bellare and Miner already pointed out, the lengths of both the secret key and signature are logarithmic in the total number of time slots.

Clearly, the scheme obtained via the binary tree certification method is less efficient than our scheme above and the random-oracle scheme of [2]. However, by properly instantiating the scheme, one can reduce its key length while maintaining its efficiency. The key observation for doing so is that we do not need the full power of ordinary signature schemes at the internal nodes, since they only need to certify two other nodes. Hence, we can use more "light-weight" schemes at these nodes, such as one-time signature schemes [8]. These are schemes which can only withstand single-message attacks, i.e. the signing key can be used only once. They are usually very efficient and have the potential for using smaller keys due to the restriction they impose on the attack. By using such schemes, we were actually able to achieve some improvements (see Appendix B), but, unfortunately, given what is currently known, this still does not seem to give us a practical implementation without random oracles.

Another way of avoiding the use random oracles in the design of forward-secure signature schemes is by using ideas of Cramer and Damgård [5]. They show how to convert a secure identification scheme of the type commit-challenge-respond (which they refer to as *signature protocols*) into a secure signature scheme without relying on random oracles. The transformation is based on the idea of authentication trees. In this model, each message has a leaf associated to it. Signing a message is simply a matter of computing the path, which they call authentication path, from the leaf associated with that message to the root. To avoid having to precompute and store the whole tree, nodes are created dynamically in a way very similar to that of the GMR scheme. And like the GMR scheme, the resulting scheme is not memoryless and needs to remember the signature of the previous message to be able to compute the next signature. The length of each signature also grows logarithmically with the number of signed messages. This can, however, be improved to give a memoryless scheme, using the same modifications that Goldreich [10] suggested for the GMR scheme. The length of each signature will now be the same, although still logarithmic in the total number of messages ever to be signed.

In the case of forward security, we would have to start with a forward-secure identification scheme (such as the one given in [2]), and then apply to it the same type of transformation described above with one main difference: we also have to account for the index of the current time period. But we can easily do so by simply replacing a message in the original case by a pair message-index in our case. Although we do not prove this result, our claim is that forward security will be preserved. The main advantage of such an approach is that we can obtain a signature scheme which is forward secure based solely on the security of the corresponding identification scheme (and thus, if we use the scheme of [2], solely on the hardness of factoring). Moreover, the lengths of both the secret and public keys are independent of the total number of time periods. Its main disadvantages are that the resulting signature scheme would be far less efficient than the one we suggest in Section 3, and would have signatures whose length is a function of the total number of signed messages (and, therefore, related to the total number of time periods).

5 Acknowledgment

We are grateful to Mihir Bellare for encouraging us to work together and for advice along the way.

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," *Advances in Cryptology – Crypto* 99 *Proceedings*, Lecture Notes in Computer Science Vol. 1666, 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*, Lecture Notes in Computer Science Vol. 1070, U. Maurer ed., Springer-Verlag, 1996.

[5] R. Cramer and I. Damgård, "Secure signature schemes based on interactive protocols," *Advances in Cryptology – Crypto* 95 *Proceedings*, Lecture Notes in Computer Science Vol. 963, D. Coppersmith ed., Springer-Verlag, 1995.

[6] R. Cramer and V. Shoup, "Signature schemes based on the Strong RSA Assumption," *Sixth Annual Conference on Computer and Communications Security*, ACM, 1999.

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

[8] S. Even, O. Goldreich, and S. Micali, "On-line/Off-line digital signatures," *Jounal of Cryptology*, Vol. 9, 1996, pp. 35–67.

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

[10] O. Goldreich, "Two remarks concerning the GMR signature scheme," *Advances in Cryptology – Crypto* 86 *Proceedings*, Lecture Notes in Computer Science Vol. 263, A. Odlyzko ed., Springer-Verlag, 1986.

[11] 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.

[12] L. Guillou and J. Quisquater, "A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory," *Advances in Cryptology – Eurocrypt* 88 *Proceedings*, Lecture Notes in Computer Science Vol. 330, C. Gunther ed., Springer-Verlag, 1988.

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

[14] C. H. Lim and P .J. Lee, "More Flexible Exponentiation with Precomputation," *Advances in Cryptology – Crypto* 94 *Proceedings*, Lecture Notes in Computer Science Vol. 839, Y. Desmedt ed., Springer-Verlag, 1994

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

[16] S. Micali, "A secure and efficient digital signature algorithm," *Technical Report MIT/LCS/TM-501*, Massachusetts Institute of Technology, Cambridge, MA, March 1994.

[17] S. Micali and L. Reyzin, "Improving the exact security of Fiat-Shamir signature schemes." In R. Baumgart, editor, *Secure Networking — CQRE [Secure] '99*, volume 1740 of *Lecture Notes in Computer Science*, pages 167–182, Springer-Verlag, 1999.

[18] K. Ohta and T. Okamoto. "A Modification of the Fiat-Shamir Scheme," *Advances in Cryptology – Crypto 88 Proceedings*, Lecture Notes in Computer Science Vol. 403, S. Goldwasser ed., Springer-Verlag, 1988, pp. 232-243.

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

A Proof of Theorem 3.3

First, we assume that if F outputs $(b,(Z,\sigma))$ as a forgery, then the hashing oracle has been queried on (b,Y,M), where $Y = Z^{2^{l(T+1-b)}}U^{\sigma}$ (any adversary can be modified to do that; this may raise the number of hash queries to $q_{\text{hash}}+1$.) We will also assume that F performs the necessary bookkeeping and does not ask the same hash query twice.[1] Note that F may ask the same signature query twice, because the answers will most likely be different.

A has to guess the time period at which F will ask the breakin query: it randomly selects i, $1 < i \le T+1$, hoping that the break-in will occur at i or later, and the forgery will be for a time-period earlier than i. It then sets $U = 1/v^{2^{l(T+1-i)}} \bmod N$ and $PK = (N,U,T)$, and runs the adversary the first time. Note that $v = S_i$ (unless $i = T+1$, in which case $v = 1/U$).

A then comes up with a random tape for F, remembers it, and runs F on that tape. A maintains two tables: a signature query table and a hash query table.

Signature queries can be answered almost at random, because A controls the hash oracle. In order to answer a signature query number s on a message M'_s during time period j'_s, A selects a random $Z_s \in Q$ and $\sigma'_s \in \{0,1\}^l$, computes $Y'_s = Z_s^{2^{l(T+1-j'_s)}}U^{\sigma'_s}$, and checks its signature query table to see if a signature query on M'_s during time period j'_s has already been asked and if Y'_s used in answering it. If so, A changes S_s and σ'_s to the Z and σ that were used in answering that query. Then A adds the entry $(s, j'_s, Y'_s, \sigma'_s, Z'_s, M'_s)$ to its signature query table and outputs $(j'_s,(Z_s,\sigma'_s))$.

Hash queries are also answered at random. To answer the t-th hashing query $(j_t,(Z_t,M_t))$, A first checks its signature query table to see if there is an entry $(s, j'_s, Y'_s, \sigma'_s, Z'_s, M'_s)$ such that $(j'_s, Z'_s, M'_s) = (j_t, Z_t, M_t)$. If so, it just outputs σ'_s. Otherwise, it picks a random $\sigma_t \in \{0,1\}^l$, records in its hash query table the quintuple $(t, j_t, Z_t, M_t, \sigma_t)$ and outputs σ_t.

Assume now the break-in query occurs during time-period $j \ge i$, and the forgery $(b,(Z,\sigma))$ is output for a time period $b < i$ (if not, or if no forgery is output, A fails). (To answer the breakin query, A outputs $v^{2^{l(j-i)}}$.) Let $Y = Z^{2^{l(T+1-b)}}U^{\sigma}$. Because we modified F to first ask a hash query on (j,Y,M), we have that, for some h, $(h,b,Y,M,\sigma) = (h, j_h, Y_h, M_h, \sigma_h)$ in the hash query table (it can't come from the signature query table, because F is not allowed to forge a signature on a message for which it asked a signature query). A finds such an h in its table and remembers it.

A now resets F with the same random tape as the first time, and runs it again, giving the exact same answers to all F's queries before the h-th hash query (it can do so because it has all the answers recorded in the tables). Note that this means that F will be asking the same h-th hash query (b,Y,X,M) as the first time. As soon as F asks the h-th hash query, however, A stops giving the answers from the tables and comes up with new answers at random, in the same manner as the first time. Let τ be the new answer given to the h.

Assume again the break-in query occurs during time-period $j' \ge i$, and the forgery $(b',(Z',\sigma'))$ is output for a time period $b' < i$.

[1] This may slightly increase the running time of F, but we will ignore costs of simple table look-up for the purposes of this analysis.

If the second forgery was not based on hash query number h, our algorithm fails. If it was, however, then we have $b = b'$ and

$$\begin{aligned} Z^{2^{l(T+1-b)}} U^{\sigma} &\equiv Z'^{2^{l(T+1-b)}} U^{\sigma'} \pmod N \Rightarrow \\ \left(v^{-2^{l(T+1-i)}}\right)^{\sigma-\sigma'} &\equiv (Z'/Z)^{2^{l(T+1-b)}} \pmod N \Rightarrow \\ v^{\sigma-\sigma'} &\equiv (Z/Z')^{2^{l(i-b)}} \pmod N \end{aligned}$$

(the last step involves taking roots of degree $2^{l(T+1-b)}$ of both sides, which we are allowed to do because both sides are in Q and remain in Q, because v is a square and $i > b$). Note that $b < i$, and σ and σ' are both at most l bits, so $\sigma - \sigma' < 2^{l(i-b)}$. By applying the lemma below (which is an algebraic generalization of Lemma 3.2), our algorithm can easily compute a square root of v, setting $\alpha = \sigma - \sigma'$, $X = Z/Z'$ and $\lambda = l(i-b)$.

Lemma A.1 Given $\alpha \neq 0, \lambda > 0, v \in Q$ and $X \in Z_N^*$ such that $v^{\alpha} \equiv X^{2^{\lambda}} \pmod N$ and $\alpha < 2^{\lambda}$, one can easily compute y s.t. $v \equiv y^2 \pmod N$.

Proof: Let $\alpha = 2^{\gamma}\beta$ where β is odd. Note that $\lambda > \gamma$. Let $\beta = 2\delta+1$. Then $\left(v^{2\delta+1}\right)^{2^{\gamma}} \equiv v^{\alpha} \equiv X^{2^{\lambda}} \pmod N$, so $v^{2\delta+1} \equiv X^{2^{\lambda-\gamma}} \pmod N$ (again, we are allowed to take roots of degree 2^{γ} because both sides are in Q). Let $y = X^{2^{\lambda-\gamma-1}}/v^{\delta} \bmod N$. Then $y^2 \equiv X^{2^{\lambda-\gamma}}/v^{2\delta} \equiv v \pmod N$. Note that it is crucial that $\alpha < 2^{\lambda}$ so that $\lambda - \gamma - 1 \geq 0$. ∎

Running Time Analysis. A runs F twice. Answering hashing and signing queries takes A no longer than it would the real oracles (ignoring the costs of table look-ups). To find which hashing query the signature corresponds to, to compute the square root of v and to finally factor N takes some exponentiations of numbers modulo N (where the exponents are at most lT bits long) and some gcd computations for division and factoring N. Thus, A exceeds the running time of F by $O(k^2 lT + k^3)$.

Probability Analysis. First, we need the following lemma.

Lemma A.2 Let $a_1, a_2, \ldots, a_{\lambda}$ be real numbers. Let $a = \sum_{\mu=1}^{\lambda} a_{\mu}$. Let $s = \sum_{\mu=1}^{\lambda} a_{\mu}^2$. Then $s \geq \frac{a^2}{\lambda}$.

Proof: Let $b = a/\lambda$ and $b_{\mu} = b - a_{\mu}$. Note that $\sum_{\mu=1}^{\lambda} b_{\mu} = \lambda b - \sum_{\mu=1}^{\lambda} a_{\mu} = a - a = 0$. Then

$$\sum_{\mu=1}^{\lambda} {a_{\mu}}^2 = \sum_{\mu=1}^{\lambda} (b - b_{\mu})^2 = \lambda b^2 - 2b \sum_{\mu=1}^{\lambda} b_{\mu} + \sum_{\mu=1}^{\lambda} b_{\mu}^2 \geq \lambda b^2 = \frac{a^2}{\lambda}.$$

∎

First, consider the probability that A's answers to F's oracle queries are distributed as those of the true oracles that F expects. This is the case unless, for some signature query, the hash value that A needs to define has already been defined through a previous answer to a hash query (call this "A's failure to pretend"). Because Z is picked at random from Z_N^*, $Z^{2^{l(T+1-j)}} U^{\sigma}$ is a random element of Q. The probability of its collision with a value from a hash query in the same execution of F is at most $(q_{\text{hash}} + 1)/|Q|$; thus, the probability (taken over only the random choices of A) of A's failure to pretend is at most $q_{\text{sig}}(q_{\text{hash}} + 1)/|Q| \leq q_{\text{sig}}(q_{\text{hash}} + 1)2^{3-k}$. This is exactly the amount by which F's probability of success is reduced because of interaction with A rather than the real signer. Let $\delta = \varepsilon - q_{\text{sig}}(q_{\text{hash}} + 1)2^{3-k}$.

Let ε_i be the probability that F produces a successful forgery and that its break-in query occurs in time-period i. Clearly, $\delta = \sum_{i=2}^{T+1} \varepsilon_i$. Assume now that A picked a specific i as the time-period for $S_i = v$. The probability of that is $1/T$.

We will now calculate the probability of the event that F outputs a valid forgery based on the same hash query both times and that the hash query was answered differently the second time and that the break-in query was i both times. Let $p_{h,i}$ be the probability that, in one run, F produces a valid forgery based on hash query number h after break-in query in time-period i. Clearly,

$$\varepsilon_i = \sum_{h=1}^{q_{\text{hash}}+1} p_{h,i}$$

Let $p_{h,i,s}$ (for a sufficiently long binary string s of length m) be the probability that, in one run, F produces a valid forgery based on hash query number h after break-in query in time-period i, given that the string s was used to determine the random tape of F and the responses to all the oracle queries of F until (and not including) the h-th hash query. We have that

$$2^m p_{h,i} = \sum_{s \in \{0,1\}^m} p_{h,i,s}.$$

Given such a fixed string s, the probability that F produces a valid forgery based on the hash query number h after break-in query in time-period i in both runs is $p_{h,i,s}^2$ (because the first forgery is now independent of the second forgery). The additional requirement that the answer to the hash query in the second run be different reduces this probability to $p_{h,i,s}(p_{h,i,s} - 2^{-l})$. Thus, the probability $q_{h,i}$ that F produces a valid forgery based on the hash query number h in both runs and that the answer to the hash query is different in the second run and that the break-in query was i in both runs is

$$\begin{aligned} q_{h,i} &= \sum_{s \in \{0,1\}^m} 2^{-m} p_{h,i,s}(p_{h,i,s} - 2^{-l}) = 2^{-m}\left(\sum_{s \in \{0,1\}^m} p_{h,i,s}^2 - 2^{-l} \sum_{s \in \{0,1\}^m} p_{h,i,s}\right) \\ &\geq \frac{2^{-m}\left(p_{h,i} 2^m\right)^2}{2^m} - 2^{-l} p_{h,i} = p_{h,i}^2 - 2^{-l} p_{h,i} \end{aligned}$$

(by Lemma A.2).

The probability that F outputs a valid forgery based on the same hash query both times and that the hash query was answered differently in the second run and that the break-in query occurred in time-period i is now

$$\sum_{h=1}^{q_{\text{hash}}+1} q_{h,i} \geq \sum_{h=1}^{q_{\text{hash}}+1} p_{h,i}^2 - \sum_{h=1}^{q_{\text{hash}}+1} 2^{-l} p_{h,i} \geq \frac{\varepsilon_i^2}{q_{\text{hash}}+1} - 2^{-l}\varepsilon_i$$

(by Lemma A.2).

Note that if this happens, then the forgery occurs in time-period $b < i$ (because the forgery has to occur before the break-in query), so A will be able to take the square root of v.

Finally, we remove the assumption that A picked i as the time-period to get the probability of A's success:

$$\varepsilon' \geq \frac{1}{T} \sum_{i=2}^{T+1} \left(\frac{\varepsilon_i^2}{q_{\text{hash}}+1} - 2^{-l}\varepsilon_i\right) \geq \frac{\delta^2}{T^2(q_{\text{hash}}+1)} - \frac{\delta}{2^l T}$$

(by Lemma A.2).

We now divide the result by 2 because only half of the choices for x will lead to a factorization of N to get the desired result. ∎

B Comparison of two tree-based schemes

In order to illustrate how one can obtain more efficient schemes based on the binary certification tree method, we compare two schemes in terms of efficiency and key size. The first is an ordinary signature scheme by Cramer-Shoup [6]. This scheme is among the most efficient schemes whose proofs of security are in the standard model. The second is a one-time signature scheme proposed in [8].

CRAMER-SHOUP SIGNATURE SCHEME. The security of this scheme is based on a variation of the RSA assumption and on a collision-free hash function (see [6] for the actual description of the scheme). The scheme has two security parameters l and l'. Reasonable values for these parameters are 160 and 512, respectively. The total length of the secret key is $2l'$. The public key has length $6l' + l$. The cost of running the key generation algorithm is approximately the cost of one modular exponentiation. Signing and verifying take each about 1.5 times the time for signing with a regular RSA algorithm. Other schemes can also be used as long as they meet the requirement of being secure against chosen-message attacks in the standard model. For example, one can choose instead the GMR signature scheme [11].

ONE-TIME SIGNATURE SCHEME. The particular one-time signature scheme we consider here is the first one of those given in [8] (which they call construction 1) based on one-way functions. It has three parameters: m, the message length; b, the block size, which is a divisor of m; and n, a security parameter for a one-way function f. Longer messages are dealt with by first hashing them into strings of length m. The hash function should be collision resistant. If we consider f to be length-preserving, then the lengths of both the secret and public keys as well as the length of a signature are $n(m+1)/b$. The key generation would require exactly $2(2^b - 1)(m/b)$ applications of f and both the signing and verification each would require $(2^b - 1)(m/b)$ applications of f. By appropriately choosing a value for b (e.g. $b = 4$), which should be logarithmic in n, one can reduce the length of the key and signature without significantly increasing the overall cost. Reasonable choices for m and n are, respectively, 128 and 55 [8], which lead to keys of length 1760 when $b = 4$ (and, hence, shorter than that of the ordinary signature scheme). By using better constructions of one-time signature schemes (e.g., Construction 3 from [8]), one can reduce the key length even further and also improve the security.

ANALYSIS. As we can see by the two examples given above, there was a small improvement on the key size when going from ordinary to one-time signature schemes. This gain, however, comes at the cost of worse security guarantees. In terms of costs, the overall picture is not so clear. It actually depends on how fast the specific one-way function being used in the one-time signature is with respect to modular exponentiation operations. For example, if the one-way function is 200 times faster, then both schemes would have about the same cost.