Composition and Efficiency Tradeoffs for Forward-Secure Digital Signatures

TAL MALKIN[*] DANIELE MICCIANCIO[†] SARA MINER[‡]

Abstract

Forward-secure digital signatures, initially proposed by Anderson in CCS 97 and formalized by Bellare and Miner in Crypto 99, are signature schemes which enjoy the additional guarantee that a compromise of the secret key at some point in time does not help forge signatures allegedly signed in an earlier time period. Consequently, if the secret key is lost, then the key can be safely revoked without invalidating previously-issued signatures. Since the introduction of the concept, several forward-secure signature schemes have been proposed, with varying performance both in terms of space and time. Which scheme is most useful in practice typically depends on the requirements of the specific application.

In this paper we propose and study some general composition operations that can be used to combine existing signature schemes (whether forward-secure or not) into new forward-secure signature schemes. Our schemes offer interesting trade-offs between the various efficiency parameters, achieving a greater flexibility in accommodating the requirements of different applications. As an extension of our techniques, we also construct the first efficient forward-secure signature scheme where the total number of time periods for which the public key is used does not have to be fixed in advance. The scheme can be used for practically unbounded time, and the performance depends (minimally) only on the time elapsed so far.

Our scheme achieves excellent performance overall, is very competitive with previous schemes with respect to all parameters, and outperforms each of the previous schemes in at least one parameter. Moreover, the scheme can be based on any underlying digital signature scheme, and does not rely on specific assumptions. Its forward security is proven in the standard model, without using a random oracle.

1 Introduction

The standard notion of digital signatures is extremely vulnerable to leakage of the secret key, which may be quite a realistic threat over the lifetime of a system. Indeed, if the secret key is compromised, any message can be forged. Consequently, not only does it invalidate all future signatures, but also all *previously issued* signatures cannot be trusted. Once a leakage has been identified, some key revocation mechanism may be invoked, but this does not solve the problem of forgeability for past signatures. Asking the signer to reissue all previous signatures is very inefficient, and, moreover, requires trusting the signer. For example, it is very easy for a dishonest signer to leak his secret

[*]AT&T Labs Research, 180 Park Avenue, Florham Park, NJ 07932, USA. E-mail: `tal@research.att.com`

[†] Department of Computer Science and Engineering, University of California, San Diego, Mail Code 0114, 9500 Gilman Drive, La Jolla, CA 92093-0114, USA. E-mail: `daniele@cs.ucsd.edu`

[‡] Department of Computer Science and Engineering, University of California, San Diego, Mail Code 0123, 9500 Gilman Drive, La Jolla, CA 92093-0123, USA. E-mail: `sminer@cs.ucsd.edu`

key in order to repudiate a previously signed document. Changing the keys very frequently is also not a practical solution, since registering new public keys and maintaining them in a place that is both publicly accessible and trusted is a difficult task.

To mitigate the consequences of possible key leaks, the notion of forward security for digital signatures was initially proposed by Anderson [1] and formalized by Bellare and Miner in [3]. The basic idea is to extend a standard digital signature algorithm with a *key update* algorithm, so that the *secret key* can be changed frequently (while the public key stays the same). The resulting scheme is forward-secure if the knowledge of the secret key at some point in time does not help forge signatures relative to some previous time period. Thus, if the secret key is compromised during time period t, then the key can be safely revoked without invalidating previously issued signatures.

Since the introduction of the concept of forward security, several such schemes have been suggested. These schemes exhibit varying performance both in terms of space (key and signature size) and time (signature, verification and update speed). Some schemes have faster signature generation and verification algorithms, but slower key generation and update procedures. Others can generate or update keys faster, but at the price of slower signing and verification. In other cases, time improvements are obtained at the cost of larger signatures or secret and public keys. Clearly, there is a trade-off between the efficiency parameters, and which scheme is most practical typically depends on the requirements of the specific application. Consider, for example, a scenario where keys are updated once a day. Then, a slower key update algorithm might be acceptable, if this helps make signature verification faster or digital signatures shorter. On the other hand, consider an electronic checkbook (e-check) application. Here the time period is not physical time, instead it corresponds to the check serial number. If your electronic wallet is stolen (with the current e-check secret key in it) you want to be able to revoke all compromised checks without invalidating the checks that were legitimately issued. This can be achieved if the secret key is updated after signing every check. In this case, fast key update is as important as fast signature generation, as the two algorithms are always used together.

1.1 Our Results

In this paper, we propose and study some general composition operations that can be used to combine *any* existing signature schemes (whether standard or forward-secure) into new forward-secure signature schemes. We use these composition operations to revisit old forward-secure digital signature schemes as well as suggest new ones that offer interesting trade-offs between the various efficiency parameters, achieving greater flexibility in accommodating different applications.

Extending our general techniques, we design a new tree-like digital signature (different from previous tree constructions) which we call the MMM scheme. This scheme is very efficient, and outperforms all previous constructions in at least some of the parameters (the price is never more than a constant increase in other parameters, and in most cases there is no price at all). For example, signing requires a single signature in the underlying scheme, and verifying requires little more than two verifications of the underlying scheme. Furthermore, the MMM scheme is the first efficient forward-secure scheme which does not require the maximal number of time periods that may be used to be fixed in advance. Instead, the user can keep calling the update procedure indefinitely. The only (theoretical) barrier to the number of time periods is exponential in the security parameter, and therefore can not be feasibly reached. Unlike all previous schemes, the efficiency parameters of the MMM scheme do not depend at all on the *maximal* number of time periods, but rather on the number of time periods elapsed so far (and this dependence is minimal).

Since our operations can use any underlying signature scheme as a building block, they can be used both to build forward-secure signature schemes based on general assumptions (say, the existence of regular digital signature schemes, which is equivalent to the existence of one-way functions [15]), and to obtain schemes based on the same number theoretic assumptions as previously used for forward-secure signatures [3, 2, 9]. Our proofs are given in the standard complexity model, based on the security of the underlying scheme, and without requiring any additional assumption.[1] In particular, no random oracle is needed (unless used by the underlying signature scheme).

1.2 Building Blocks / Related Work

In recent years, several forward-secure signature schemes have been proposed in the literature. Some of the first solutions described in [1] had the undesirable property that keys or signature size were linear in the (maximal) number of time periods T. (For example, one can generate T different secret/public key pairs using a conventional signature scheme, and delete a secret key from memory each time the update algorithm is invoked.) Subsequent efforts concentrated on finding schemes where the size parameters (key and signature size) were independent of, or at most logarithmic in, the number of time periods, possibly at the price of slower signing, verifying or update algorithms. (E.g., the solutions proposed by Bellare and Miner [3] and Abdalla and Reyzin [2] had signing and verification time linear in T.) Here, "size independent of T" means that the size depends only on the value of the security parameters. However, it should be noted that the security parameters must be superlogarithmic in T to avoid exhaustive search attacks, so having no explicit dependency on $\log T$ does not necessarily give shorter keys. For a fair comparison of different schemes, the actual dependency of the space and time complexity of all algorithms on the security parameters must be taken into account.

In this section, we briefly highlight these dependencies for all previously proposed schemes. These schemes can be divided into two categories: those which are generic constructions built using any standard signature scheme as a black box, and those based on specific number theoretic assumptions, e.g., hardness of factoring large integers. We use two different security parameters in describing the efficiency of these schemes:

- l: a security parameter such that exhaustive search over l-bit strings is infeasible. This is the security parameter of conventional (symmetric) cryptographic operations, e.g., the seed length of pseudo-random generators, or the output length of cryptographic hash functions.

- k: a security parameter such that k-bit numbers are hard to factor, or more generally common number theoretic problems (such as inverting the RSA function) become infeasible.

It is important to distinguish between the two values because factoring can be solved in sub-exponential time $(\exp(k^{1/3}\log^{2/3} k))$, so asymptotically one needs $k \approx O(l^3)$ to be much bigger than l. In practice, acceptable values of the security parameters are $k = 1000$ and $l = 100$. In the analysis of all constructions we assume that modular multiplications of k-bit numbers are performed in k^2 time and hash functions and pseudo-random generators run in l^2 time. For $k = 1000$ and $l = 100$ this means that a block cipher application is 100 times faster than modular arithmetic operation.

In the tables below, we omit some of the lower order terms from the running time, e.g., if an algorithm requires one modular exponentiation plus one hashing, we simplify the running time expression $O(k^3) + O(l^2)$, and write only the most significant term $O(k^3)$. The only assumptions used in these simplifications are that $\log T = o(l)$ and $l = o(k)$.

[1] We also use pseudo-random generators [4, 16] and universal one way (a.k.a., target collision resistant) hash functions [13], but these are known to be equivalent to the existence of digital signature schemes.

Generic constructions

Generic constructions are those proposed by Anderson [1], the tree scheme of Bellare and Miner [3], and the Krawczyk variant of the Anderson scheme with reduced secret storage [10]. Their performance is summarized in in Figure 1. As usual, T stands for the number of time periods in the scheme. When a standard signature scheme is required, we assume key generation, signing and verifying can all be performed in $O(lk^2)$ time, and public keys, secret keys and signatures are $O(k)$ bits long. For example, this is the performance of the signature schemes of Guillou and Quisquater [8], and of Micali [12]. Note that we have broken up the storage required of the signer into "Secret key size" and "Non-secret storage", to better illustrate the differences between these schemes. This was indeed the main improvement of Krawczyk scheme, which otherwise is essentially the same as the one originally proposed by Anderson. However, it should be remarked that even the non-secret storage must still be secure in the sense that it must be stored in (publicly readable) tamper-proof memory. Changing or altering the non-secret storage would disrupt the signing algorithm, making signature generation impossible. For simplicity, in the rest of this paper we will drop the distinction between secret and non-secret storage.

	Anderson	Binary Tree (BM)	Krawczyk
Key gen time	lk^2T	$3lk^2 \log T$	$2lk^2T$
Signing time	lk^2	lk^2	lk^2
Verification time	$2lk^2$	$lk^2 \log T$	$2lk^2$
Key update time $(*)$	$O(1)$	$2lk^2$	lk^2
Secret key size	kT	$k \log T$	k
Non-secret storage	$2kT$	$4k \log T$	kT
Public key size	k	k	k
Signature size	$3k$	$2k \log T$	$3k$

Figure 1: Comparing parameters of generic constructions [1, 3, 10]. $(*)$The running time of the Binary Tree update algorithm is amortized over all update operations. If no amortization is used, the worst running time can be bigger by a factor $\log T$.

As shown in the table, the Binary Tree scheme has much faster key generation, and smaller secret/non-secret key storage than Anderson and Krawczyk schemes, with the linear dependency on T replaced by a logarithmic function. In exchange, the price paid is a slower verification procedure and longer signatures, which both increase by a factor $\log T$. For a detailed description of the schemes the reader is referred to the original papers [1, 3, 10].

Constructions based on specific security assumptions

Number theoretic schemes were proposed by Bellare and Miner [3], Abdalla and Reyzin [2], and very recently (and independently of our work), by Itkis and Reyzin [9]. These are proven to be secure in the random oracle model, assuming, roughly, that factoring is hard (for [3, 2]) or that taking any root modulo a composite is hard (for [9]). The performance of these schemes is summarized in table 2.

Except for the key update time, the Itkis-Reyzin (IR) scheme is essentially optimal among the number theoretic schemes. However, key update is very slow (linear in T) making it impractical for some applications (e.g., the e-check application described in the introduction). Itkis and Reyzin [9] also suggest a variant of their scheme where the update time is only proportional to $\log T$, but

	BM	AR	IR
Key gen time	lk^2T	lk^2T	$k^5 + (k + l^3)lT$
Signing time	$(T + l)k^2$	lk^2T	$2k^2l$
Verification time	$(T + l)k^2$	lk^2T	$2k^2l$
Key update time	lk^2	lk^2	$(k^2 + l^3)lT$
Secret key size	lk	k	$2k$
Public key size	lk	k	$2k$
Signature size	$2k$	k	k

Figure 2: Comparing parameters of schemes built on specific security assumptions [3, 2, 9].

the secret key size increases by a factor $\log T$, thus matching generic constructions like the binary tree scheme. (In fact, generic construction have potentially smaller keys because they can be based on any signature scheme with possibly shorter keys than factoring based ones.) Interestingly, the seemingly "optimal" (i.e., independent of T) verification time of the IR scheme, is not necessarily better than other schemes in which this time is proportional to $\log T$. For example, by properly instantiating the binary tree scheme with a basic signature algorithm with fast verification procedure (e.g., Rabin's signature scheme [14] which has verification time k^2), one can obtain a forward-secure signature scheme where verification takes only $O(k^2 \log T)$, beating the $O(lk^2)$ running time of the IR scheme because $\log T = o(l)$. We will see in a later section that even faster verification times are possible, using a different tree construction.

Comparison to Our Results

We start by showing a general composition operations that can be applied to any underlying scheme, yielding new forward-secure schemes. These operations may in fact be viewed as generalizations of the generic schemes of Bellare-Miner [3] and Krawczyk [10]. In Section 4 we provide some simple instantiations and their parameters in order to demonstrate the large range of combinations and the possible trade-offs. These operations, however, should be viewed as general tools that can be instantiated based on the needs and resources of the application (e.g., the underlying protocols that are available, which parameters are most critical, which hardness assumptions are considered better, and so on).

All previously known schemes (with the only exception of the very inefficient "long signature scheme" described in [3]) require the total (or maximal) number of time periods T to be fixed in advance and passed as a parameter to the key generation algorithm. The value of T then contributes to the overall performance of these schemes (at least proportionally to $\log T$ and a security parameter, but in most cases linearly for at least some parameters). We propose a new scheme whose performance avoids any dependence on T, and depends only the number of time periods elapsed so far (and even this dependence is minimal). In fact, T need never be fixed. The new scheme, called MMM, is very competitive compared with all previous schemes, and its specific parameters are detailed in Section 5.

As can be seen, in addition to the novel aspects which are not shared by any previous scheme, our schemes also combine the best of both types of previous constructions in terms of efficiency and security. Indeed, we construct schemes based on generic assumptions, yet we achieve competitive performance and stronger security guarantees even when compared to the best previous schemes which used specific number theoretic assumptions.

2 Preliminaries

2.1 Definitions and Notation

In this section, we define key-evolving and forward-secure signature schemes, in the style of [3].

A *key-evolving* signature scheme $\mathcal{S}$ consists of four algorithms: a key generation algorithm *KeyGen*, a secret key evolution algorithm *Update*, a signature generation algorithm *Sign*, and a signature verification algorithm *Verify*. It differs from a standard digital signature scheme in that the secret key is subject to an update (or evolution) algorithm, and the time for which the scheme is in use is divided into *time periods*. The public and secret keys are denoted pk and sk respectively, and the secret key sk changes via each invocation of the key update algorithm. We write $sk^{(j)}$ when we want to emphasize that a particular value of the secret key is relative to the jth time period. It is important to notice that the public key for the scheme remains constant throughout, and each signature is verified using the same public key pk, regardless of which $sk^{(j)}$ was used to generate it.

For a key-evolving signature scheme to be *forward-secure*, it must be computationally infeasible for an adversary, even after learning the secret key of the scheme for a particular time period, to forge a signature on a new message for a time period earlier than that for which it possesses the secret key.

Typically, the total number of time periods T for a particular instantiation of a forward-secure signature scheme must be fixed in advance, and passed as a parameter to the key generation algorithm *KeyGen*. Moreover, T is usually included in the public key pk, secret keys $sk^{(j)}$, and signatures σ, as it is required by the update, signing and verification algorithms. The secret keys $sk^{(j)}$ and signatures σ also include the current or issuing time period t. In this paper, we use the convention that both T and t are passed as *external* parameters to *Update*, *Sign* and *Verify*, instead of being included in the key and signature strings. This is to avoid unnecessary duplications when signature schemes are combined using our composition operations. However, it should be noted that most update, sign and verify algorithms need T and t to work properly, and therefore these values should be thought as integral parts of keys and signatures. Below, we summarize the input and output parameters of each algorithm of a forward-secure digital signature scheme:

- The key generation algorithm *KeyGen* takes as input the total number of time periods T, and outputs a pair $(pk, sk^{(0)})$ of public and secret keys.
- The key update algorithm *Update* takes as input T, the current time period t, and the secret key $sk^{(t)}$ for time period t, and changes $sk^{(t)}$ into $sk^{(t+1)}$. If $t+1 = T$ then *Update* completely erases the secret key sk, and returns the empty string.
- The signing algorithm *Sign* takes as input T, the current time period t, the corresponding secret key $sk^{(t)}$, and a message to be signed M, and outputs a signature (σ, t).
- The verification algorithm *Verify* takes as input T, pk, M, σ, and t, and accepts if and only if σ is a valid signature tag for message M and secret key $sk^{(t)}$. Note that here t represents the time period during which σ was issued, which is not necessarily the current one.

Additionally, all algorithms implicitly take as input one or more security parameters.

2.2 Adversary Model

We use the adversary model established by Bellare and Miner [3], which is an extension of the notion of Goldwasser, Micali, and Rivest [7]. The security property we seek is that, even after breaking into the signer's machine and learning the secret key for time period b, the forger F

should be unable to generate signatures for time periods before the break-in time b. Consequently, the model incorporates three phases of execution for the forger. After receiving the public key for the scheme, F starts in the *chosen message attack phase*, where it can query a signing oracle on messages of its choice during time periods in order from earliest to latest. In the *break-in phase*, F learns the secret signing key for a time period b of its choice. Finally, in the *forgery phase*, F outputs a message-signature pair.

The forger is considered successful in breaking the scheme if, in its *forgery phase*, it outputs a message-signature pair (M, σ), for some time period f, such that all of the following hold:

- $\mathit{Verify}(T, pk, M, \sigma, f)$ accepts. That is, the forgery looks valid.
- F did not query the signing oracle on message M during time period f. That is, the forgery is on a "new" message.
- The forgery time f is strictly *earlier* than the time period b in which F broke in and learned the secret key of the scheme. (Generating a forgery for the break-in time period b or any subsequent time period is trivial once the secret key for time b is learned, and hence is not considered a successful forgery.)

In practice, we assume that the forger runs by making a series of requests for action by its signing oracle. The scheme starts in time period $j = 0$, and the forger is given the public key for the scheme. Then, the forger's requests are handled in order. Each request is one of the following three types:

- Update time period: Call $\mathit{Update}(T, t, sk^{(t)})$, where t is the current time period.
- Signature query on message M: Signing oracle returns $\mathit{Sign}(T, t, sk^{(t)}, M)$, a signature of M during the current time period t.
- Break in: Forger F is given $sk^{(b)}$, the secret key for the current time period b.

Without loss of generality, we can assume that when F requests a break-in, it will make no further requests. Once it possesses the secret key for the current time, it can update the secret key as much as it wishes, and can generate signatures on messages of its choice in any time period.

3 Composition Methods

In this section we describe some general methods that can be used to combine both standard and forward-secure signature schemes into forward-secure schemes with more and more time periods. In order to unify the presentation, we regard standard signature schemes as forward-secure signature schemes with one time period, namely $T = 1$. Let $\Sigma_0 = (\mathit{KeyGen}_0, \mathit{Update}_0, \mathit{Sign}_0, \mathit{Verify}_0)$ and $\Sigma_1 = (\mathit{KeyGen}_1, \mathit{Update}_1, \mathit{Sign}_1, \mathit{Verify}_1)$ be two forward-secure signature schemes with T_0 and T_1 time periods respectively. We consider two methods to combine Σ_0 and Σ_1 into a new forward-secure signature scheme Σ with a larger number of time periods T. The first composition method results in a new scheme $\Sigma = \Sigma_0 \oplus \Sigma_1$ with $T = T_0 + T_1$ time periods. The second composition method results in a new scheme $\Sigma = \Sigma_0 \otimes \Sigma_1$ with $T = T_0 \cdot T_1$ time periods. We call these composition operations the "sum" and the "product" constructions, respectively. We analyze these schemes and some instantiations (including iterative application of the same composition operation) in following sections.

The sum and product procedures make use of specialized versions of the key generation algorithms KeyGen_i $(i = 0, 1)$ that produce only the *secret* or the *public* key of the original schemes.

Algorithm $KeyGen(r)$: $(r_0, r_1) \leftarrow G(r)$ $(sk_0, pk_0) \leftarrow KeyGen_0(r_0)$ $pk_1 \leftarrow PKeyGen_1(r_1)$ $pk \leftarrow \mathrm{Hash}(pk_0, pk_1)$ Return $(\underbrace{\langle sk_0, r_1, pk_0, pk_1 \rangle}_{sk}, pk)$	Algorithm $Update(t, \overbrace{\langle sk', r_1, pk_0, pk_1 \rangle}^{sk})$ If $(t + 1 < T_0)$ Then $sk' \leftarrow Update_0(t, sk')$ Else If $(t + 1 = T_0)$ Then $sk' \leftarrow SKeyGen_1(r_1)$; $r_1 \leftarrow 0$ Else $sk' \leftarrow Update_1(t - T_0, sk')$ End
Algorithm $Sign(t, \overbrace{\langle sk', r_1, pk_0, pk_1 \rangle}^{sk}, M)$ If $(t < T_0)$ Then $\sigma' \leftarrow Sign_0(t, sk', M)$ Else $\sigma' \leftarrow Sign_1(t - T_0, sk', M)$ Return $(\underbrace{\langle \sigma', pk_0, pk_1 \rangle}_{\sigma}, t)$	Algorithm $Verify(pk, M, \overbrace{\langle \sigma', pk_0, pk_1 \rangle}^{\sigma}, \mathrm{t})$ If $(\mathrm{Hash}(pk_0, pk_1) = pk)$ Then If $(t < T_0)$ Then $Verify_0(pk_0, M, \sigma, t)$ Else $Verify_1(pk_1, M, \sigma, t - T_0))$ Else Reject

Figure 3: *The algorithms defining the sum composition.*

These specialized versions are called $SKeyGen_i$ and $PKeyGen_i$ respectively. As we shall see, the diversification of *KeyGen*, *PKeyGen* and *SKeyGen* plays a critical role in the analysis of forward-secure signature schemes obtained by the iterated application of the basic composition operations.

3.1 The "Sum" Composition

Let $\Sigma_0 = (KeyGen_0, Update_0, Sign_0, Verify_0)$ and $\Sigma_1 = (KeyGen_1, Update_1, Sign_1, Verify_1)$ be two forward-secure signature schemes with T_0 and T_1 time periods respectively. Also, let $PKeyGen_0$, $PKeyGen_1$, $SKeyGen_0$, $SKeyGen_1$ be specialized versions of their key generation algorithms that produce either the public or the secret key. We define a new scheme $\Sigma = \Sigma_0 \oplus \Sigma_1$ (called the sum of Σ_0 and Σ_1) with $T = T_0 + T_1$ time periods. The public key for the sum scheme pk is the (collision-resistant) hash of the public keys pk_0, pk_1 of the two constituent schemes. The composition works by first expanding a random seed r into a pair of seeds (r_0, r_1) using a length-doubling pseudorandom generator, and generating keys for both Σ_0 and Σ_1 using pseudorandom seeds r_0 and r_1 respectively. Then, the secret key for the Σ_1 scheme is deleted, while its public key and the randomness r_1 are saved. (Deleting the second secret key, and later recomputing it using the seed r_1, is essential to keep the size of the secret key small when the composition operation is iterated many times.) Signatures are generated using secret keys from the Σ_0 scheme during the first T_0 time periods. Then the Σ_1 key generation process is run again with the random string r_1, to produce the same secret key obtained earlier. (Notice that the public key will also be the same.) From this point forward, the Σ_0 secret key is deleted and only the Σ_1 keys are used. Signatures during any time period include the public keys for both schemes, so that the tag generated can be checked against the relevant public key, and so the authenticity of the signature can be checked against the hash value of both public keys, which was published. The details of the scheme are given in Figure 3.

Note that the update algorithm is usually very fast, but its worst case can be quite slow, as it includes key generation for Σ_1. This can be amortized, and, at a small price, made uniformly

Algorithm $\mathit{KeyGen}(r)$ $(r_0, r_1) \leftarrow G(r)$ $(r_1', r_1'') \leftarrow G(r_1)$ $(\mathit{sk}_0, \mathit{pk}) \leftarrow \mathit{KeyGen}_0(r_0)$ $(\mathit{sk}_1, \mathit{pk}_1) \leftarrow \mathit{KeyGen}_1(r_1')$ $\sigma \leftarrow \mathit{Sign}_0(0, \mathit{sk}_0, \mathit{pk}_1)$ $\mathit{sk}_0 \leftarrow \mathit{Update}_0(0, \mathit{sk}_0)$ Return $(\underbrace{\langle \mathit{sk}_0, \sigma, \mathit{sk}_1, \mathit{pk}_1, r_1'' \rangle}_{\mathit{sk}}, \mathit{pk})$	Algorithm $\mathit{Update}(t, \overbrace{\langle \mathit{sk}_0, \sigma, \mathit{sk}_1, \mathit{pk}_1, r \rangle}^{\mathit{sk}})$ If $(t + 1 \neq 0 \bmod T_1)$ Then $\mathit{Update}_1(t \bmod T_1, \mathit{sk}_1)$ Else $(r', r) \leftarrow G(r)$ $(\mathit{sk}_1, \mathit{pk}_1) \leftarrow \mathit{KeyGen}_1(r')$ $\sigma \leftarrow \mathit{Sign}_0(\lfloor \frac{t}{T_1} \rfloor, \mathit{sk}_0, \mathit{pk}_1)$ $\mathit{sk}_0 \leftarrow \mathit{Update}_0(\lfloor \frac{t}{T_1} \rfloor, \mathit{sk}_0)$ End
Algorithm $\mathit{Sign}(t, \overbrace{\langle \mathit{sk}_0, \sigma_0, \mathit{sk}_1, \mathit{pk}_1, r \rangle}^{\mathit{sk}}, M)$ $\sigma_1 \leftarrow \mathit{Sign}_1(\mathit{sk}_1, M, t \bmod T_1)$ Return $(\underbrace{\langle \mathit{pk}_1, \sigma_0, \sigma_1 \rangle}_{\sigma}, t)$	Algorithm $\mathit{Verify}(\mathit{pk}, M, \overbrace{\langle \mathit{pk}_1, \sigma, \sigma' \rangle}^{\sigma}, t)$ $v_0 \leftarrow \mathit{Verify}_0(\mathit{pk}, \mathit{pk}_1, \sigma, \lfloor \frac{t}{T_1} \rfloor)$ $v_1 \leftarrow \mathit{Verify}_1(\mathit{pk}_1, M, \sigma', t \bmod T_1)$ Return $(v_0 \wedge v_1)$

Figure 4: *The algorithms defining the product composition.*

small. In this case, the update algorithm will be somewhat more complex, as partial computation of a future update will be performed at each stage. See the discussion in Section 3.3.

3.2 The "Product" Composition

Let $\Sigma_0 = (\mathit{KeyGen}_0, \mathit{Update}_0, \mathit{Sign}_0, \mathit{Verify}_0)$ and $\Sigma_1 = (\mathit{KeyGen}_1, \mathit{Update}_1, \mathit{Sign}_1, \mathit{Verify}_1)$ be two forward-secure signature schemes with T_0 and T_1 time periods respectively. As usual, let $\mathit{PKeyGen}_0$, $\mathit{PKeyGen}_1$, $\mathit{SKeyGen}_0$, $\mathit{SKeyGen}_1$ be specialized versions of their key generation algorithms. We define a new scheme $\Sigma = \Sigma_0 \otimes \Sigma_1$ (called the product of Σ_0 and Σ_1) with $T = T_0 \cdot T_1$ time periods.

The idea is to combine signatures of Σ_0 and Σ_1 in a chaining construction. In the process, we will generate several instances of the Σ_1 scheme, one for every time period of Σ_0, so to achieve forward security. Specifically, for each time period in the Σ_0 scheme, which we will call an *epoch*, an instance of the Σ_1 scheme is generated, and the public key of the Σ_1 scheme is signed using the Σ_0 scheme. The public key of the entire scheme is simply the public key of the Σ_0 scheme, and a signature on a particular message M includes the signing time period, the public key of an instantiation of the Σ_1 scheme, the Σ_0 scheme signature on the Σ_1 public key, and the Σ_1 signature on the message. The scheme is precisely defined in Figure 4.

3.3 Performance Analysis

In this subsection we relate the size and running time performance of the sum and product constructions to those of their components. All relations are easily derived just inspecting the code in Figures 3,4.

Theorem 1 *Let* $\mathrm{SK_i}$, $\mathrm{PK_i}$ *and* $\mathrm{SIG_i}$ *be the secret key, public key and signature sizes of forward-secure signature algorithms* Σ_i *for* $i = 0, 1$. *Then, the key and signature sizes of the sum and product schemes are:*

- $\Sigma_0 \oplus \Sigma_1$:
  - $\mathrm{PK} = l$
  - $\mathrm{SK} = \max(\mathrm{SK}_0, \mathrm{SK}_1) + \mathrm{PK}_0 + \mathrm{PK}_1 + l$
  - $\mathrm{SIG} = \max(\mathrm{SIG}_0, \mathrm{SIG}_1) + \mathrm{PK}_0 + \mathrm{PK}_1$
- $\Sigma_0 \otimes \Sigma_1$:
  - $\mathrm{PK} = \mathrm{PK}_0$
  - $\mathrm{SK} = \mathrm{SK}_0 + \mathrm{SK}_1 + \mathrm{PK}_1 + \mathrm{SIG}_0 + l$
  - $\mathrm{SIG} = \mathrm{SIG}_0 + \mathrm{SIG}_1 + \mathrm{PK}_1$

Theorem 2 *Let* $\mathrm{KG_i}$, $\mathrm{SG_i}$, $\mathrm{VF_i}$ *and* $\mathrm{UP_i}$ *be the key generation, signature, verification and (amortized) key update time of forward-secure signature algorithms* Σ_i *for* $i = 0, 1$. *Then the running times of the sum and product schemes are:*

- $\Sigma_0 \oplus \Sigma_1$:
  - $\mathrm{KG} = \mathrm{KG}_0 + \mathrm{KG}_1 + l^2$
  - $\mathrm{SG} = \max\{\mathrm{SG}_0, \mathrm{SG}_1\}$
  - $\mathrm{VF} = \max\{\mathrm{VF}_0, \mathrm{VF}_1\} + l^2$
  - $\mathrm{UP} = (\mathrm{SKGtime}_1 + (T_0 - 1)\mathrm{UP}_0 + (T_1 - 1)\mathrm{UP}_1)/(T_0 + T_1 - 1)$
- $\Sigma_0 \otimes \Sigma_1$:
  - $\mathrm{KG} = \mathrm{KG}_0 + \mathrm{KG}_1 + \mathrm{SG}_0 + \mathrm{UP}_0 + 2l^2$
  - $\mathrm{SG} = \mathrm{SG}_1$
  - $\mathrm{VF} = \mathrm{VF}_0 + \mathrm{VF}_1$
  - $\mathrm{UP} = (T_0(T_1 - 1)\mathrm{UP}_1 + (T_0 - 1)(l^2 + \mathrm{KG}_1 + \mathrm{SG}_0 + \mathrm{UP}_0))/(T_0T_1 - 1)$
    $= \mathrm{UP}_1 + \frac{T_0 - 1}{T_0T_1 - 1}(l^2 + \mathrm{KG}_1 + \mathrm{SG}_0 + \mathrm{UP}_0 - \mathrm{UP}_1)$
    $\leq \mathrm{UP}_1 + (l^2 + \mathrm{KG}_1 + \mathrm{SG}_0 + \mathrm{UP}_0 - \mathrm{UP}_1)/T_1$

Amortized Update Algorithm. While the amortized update time in the sum construction is quite good, the worst case update time can be very bad (linear in T). This bad case happens when the left child is finished (at time T_0), and a new secret key for the right child needs to be generated. To make the update procedure uniformly fast, we can distribute the computation of this key over the time periods of the left child, so that by the time the left child is finished, the key for the right child is ready. Distributing the work (time) is straight-forward, but space becomes an issue, since the intermediate results of the computation need to be kept. So, the space necessary to generate the secret key for the right child will be added to size of the secret key of the scheme. Fortunately, when composing the sum algorithm with itself, the size of the initial secret key for the right child will be logarithmic, so the update amortization can be done efficiently, with only a small constant factor in the secret key size (as it requires a $\log T$ additive factor to the secret key). Details will be given in the final version of the paper.[2]

[2]Interestingly, a similar (standard) pebbling technique has also been independently used by [9] to amortize the update time in their scheme. They then proceed to use a more sophisticated and elegant pebbling technique which allows them to further improve their performance by a factor of 2.

4 Constructions Using Our Composition Methods

In this section, we demonstrate how different performance tradeoffs can be exploited in selecting a forward-secure signature scheme, and how our composition operations can be used, on their own or in combination, to achieve schemes with a wide array of properties. In addition to schemes quite similar to previously-known generic constructions, we describe new schemes obtained via our composition operations, and compare them to existing alternatives. Another scheme which uses a clever combination of the sum and product operations will be described separately in the next section.

4.1 The BM $\otimes$ BM scheme

We begin with a simple example that illustrates the tradeoff between time and space parameters. Consider the main scheme of Bellare and Miner with T time periods, henceforth denoted $\mathrm{BM}(T)$. The signatures produced by this scheme have size $2k$, but signing and verifying time are proportional to T. On the other hand, using our product composition, we can construct scheme $\mathrm{BM}(\sqrt{T}) \otimes \mathrm{BM}(\sqrt{T})$, which also has T time periods. But, using this construction with the same security parameter values, in exchange for a signature whose size grows to $4k + lk$ and a secret key whose size is $2k + 3lk + l$, signing time becomes proportional to $\sqrt{T}$, and verifying takes $2\sqrt{T}$ time. This is a significant savings in signing and verification time, in exchange for a relatively small increase in key and signature size. Additionally, we point out that, although some particular updates in the product composition are longer than others, the computation can be amortized (as discussed in Section 3.3) so that the cost per time period is constant.

4.2 The Iterated Product Construction $\mathbf{S}^{\otimes}_{\log\log \mathbf{T}}$

Starting with any standard (non-forward-secure) signature scheme S, we can use the sum composition once to get a forward-secure scheme $S_1 = S \oplus S$, with two time periods. We can then iterate the product composition in the following way to get a scheme with T time periods. Using S_1 as our base scheme, we define a series of schemes $S_2, \ldots, S_{\log\log T}$ such that $S_i = S_{i-1} \otimes S_{i-1}$. The final scheme, $S^{\oplus}_{\log\log T}$ has $2^{2^{\log\log T}} = T$ time periods.

The analysis given in Section 3.3 allows us to compute the performance of the scheme. The public key of the scheme is simply one hash value, but the secret key grows to be $\log\log T(2\mathrm{SK_S} + 6\mathrm{PK_S} + \mathrm{SIG_S} + 4l)$. The signature size increases to $2\mathrm{SIG_S} + 4\mathrm{PK_S} + l$. While key generation time has increased to $\log\log T(5\mathrm{KG_S} + \mathrm{SG_S} + 2l^2 + l)$, signing is as fast as with the original scheme S. Verification time increases by a factor of only $2\log\log T$ plus one hash computation. Update, once amortized, takes less than $3\mathrm{KG_S} + \mathrm{SG_S} + l^2 + l$ time.

We note that this construction, in the end, is nearly exactly the binary tree construction given in [3], as there is a signature at each node in the tree. When this scheme is instantiated with the Rabin signature scheme [14], where verifying is simply a squaring operation, it is clear that verification for the new scheme will be faster than even the very efficient scheme of [9]. Notice that for all internal nodes, one-time signatures (e.g., those of [5]) can be used instead of general digital signatures. See also [2] for a general discussion about using one-time signatures in the context of forward-security.

4.3 The Iterated Sum Construction $\mathbf{S}^{\oplus}_{\log \mathbf{T}}$

In this example, we begin with any standard (non-forward-secure) signature scheme S (viewed as a forward-secure scheme with only one time period). We can then iterate our sum composition

$\log T$ times to generate a new scheme with T time periods as follows. Specifically, we define a series of schemes $S_0, S_1, \ldots, S_{\log T}$ such that $S_0 = S$ and $S_i = S_{i-1} \oplus S_{i-1}$. $S_{\log T}$, has $2^{\log T} = T$ time periods, and is the scheme we consider here.

Again, we refer to the performance analysis given in Section 3.3. The public key for this scheme is simply a hash value, and the secret key is $\log T$ secret keys of the underlying scheme S, as well as 2 public keys from S and $3 \log T$ hash values. Relative to the underlying scheme S, the size of a signature increases by only $2 \log T$ hash values, and key generation takes T times longer plus T hash computations. Signing time is the same as with S, while verifying time increases by $l^2 \log T$. Note that in the iterated sum construction verification is much faster and signatures are shorter than the iterated product scheme, since the certification tree uses hashes only, rather than digital signatures at each node. Finally, the update time can be amortized as discussed in Section 3.3.

We note that this scheme can be described using two binary trees, where a length-doubling pseudorandom generator is used to generate the secret keys in a tree-like manner similar to the pseudorandom function construction of [6], and a hash tree (a la Merkle [11]) is used to certify the corresponding public keys (thus yielding a diamond-shaped construction). The idea of using a hash-based certification tree for public keys is not new; it was described as a variant of the main construction in [10]. The scheme in [10] also uses the idea of a pseudorandom generator in order to save storage space for the signer, but uses it in a linear way, rather than in our tree-based fashion.

4.4 The $\mathbf{IR}_{10}^{\oplus}$ Scheme

Here, we begin the Itkis-Reyzin scheme in [9] (which we denote $\mathrm{IR}(T)$) as our base scheme. By applying the sum construction in an iterated way (as described in Scheme 4.3) only 10 times, we construct a scheme with a factor of 1024 more time periods. Signing using this construction would take exactly the same time as in the original scheme, the (amortized) update procedure would be slowed down by a factor of less than 4, and the public key size would be decrease by a factor of 20 for typical parameter values. In contrast, using $\mathrm{IR}(1024T)$ instead would result in a slowdown factor of 1024 in the update procedure and a public key of the same size as $\mathrm{IR}(T)$. Illustrating the tradeoff existing among the parameters, though, in our scheme, the signature size would triple, the secret key size would increase by a factor of 12, and verification time would increase by ad *additive* term $10l^2$, whereas in the IR scheme, these remain constant as the number of time periods grows. Both our scheme and the IR scheme have a key generation algorithm linear in the number of time periods.

5 The MMM Scheme

In this section we present a new scheme, which we call MMM, that is obtained by the iterated application of the sum composition operation together with an asymmetric variant of the product construction. The main feature of the MMM scheme is that the number of time periods is essentially unbounded: the number of available time periods is limited only by the security offered by security parameter l, i.e., we cannot use more than 2^l time periods because otherwise the scheme can be broken. Moreover, the scheme exhibits excellent performance, with almost instantaneous key generation, and update, signing and verification speed that depend on the *current* time period t, instead of being functions of the *maximum* time period T. This way, the performance of the MMM scheme (in terms of signing, verifying and update operations) in the initial time periods is the same as if we had chosen a small bound on T. Only if a large number of time periods is used will the performance slightly degrade, still remaining competitive with all other schemes where T must be fixed in advance.

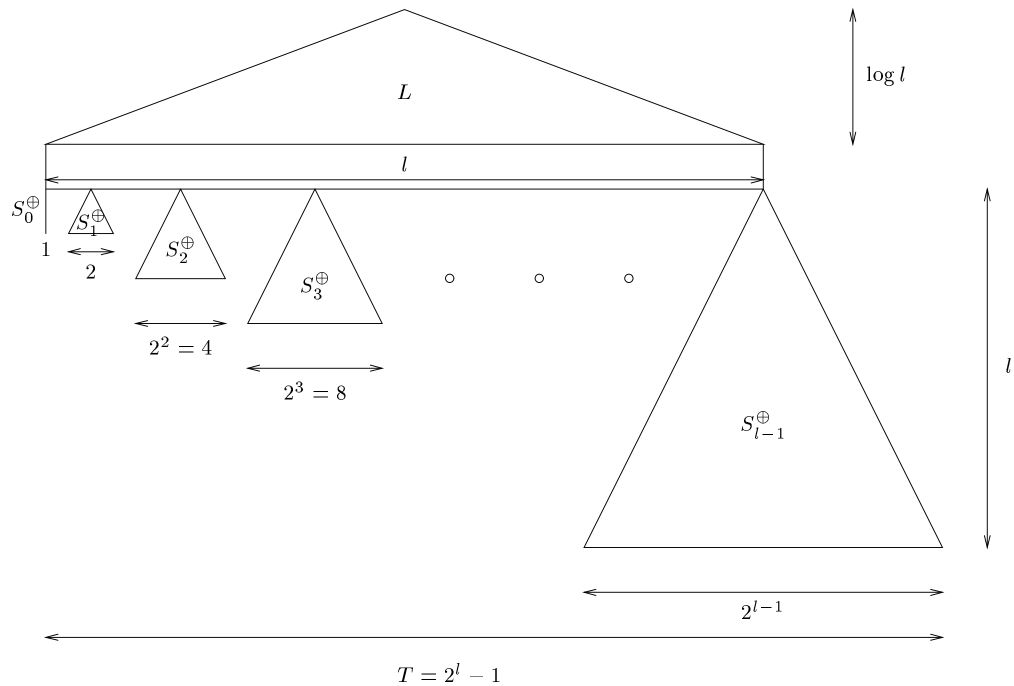

Figure 5: The asymmetric product construction.

The idea is the following. We start from a regular, non-forward-secure, digital signature scheme S and build a forward-secure signature scheme L with l time periods, iterating the sum composition $\log l$ times. In other words, we use the iterated sum scheme $L = S_{\log l}^{\oplus}$ from section 4.3. We then take the product of L with another scheme of the form $S_i^{\oplus}$, but with a twist (see later). Remember, in the product construction we build a tree where the top part is given by an instance of the L scheme, and at every leaf of L we attach an instance of the $S_i^{\oplus}$ scheme. The twist here is that we use a different i for every leaf (see figure 5), $S_0^{\oplus}$ for the first epoch, $S_1^{\oplus}$ for the second, and so on up to $S_{l-1}^{\oplus}$ for the last epoch. We see that the total number of available time periods is

$$T = \sum_{i=0}^{l-1} 2^i = 2^l - 1$$

that is, practically unbounded, because 2^l must be much bigger than T to avoid exhaustive search attacks.

We now analyze the performance of the MMM scheme with respect to various efficiency parameters.

Signature generation time. Signing a message only requires computing a signature using the secret key at the leaf node corresponding to the current time period. So, signing is as efficient as regular signature schemes.

Signature size. During time period t, each signature consists of $\log l$ hash values, a public key and a digital signature for the top level tree, and $\log t$ hash values, a public key and a digital signature for the bottom level tree. So, the total size of the signature is $4k + (\log l + \log t)l$ bits. For typical values of the security parameters, this exceeds the length k of a regular digital signature only by a constant factor.

Verification time. Verification consists of 2 regular signature verifications, and $\log l + \log t$ hash function evaluations. If the Guillou-Quisquater [8] signature scheme is used, this is $4k^2l + (\log l + \log t)l^2 = O(k^2l)$, i.e., just twice as much as the regular signature scheme.

Public key size. Public key is just a hash value: l bits.

Secret key size. The secret key has roughly the same size as a digital signature: $O(k + (\log l + \log t)l)$ bits.

Update time. The update operation is the trickiest part of this construction and it is a good example with which to illustrate how the cost of key updating can be amortized across many time periods. In this case the most expensive update operations are those in between epochs: when we finish using the subtree S_i relative to epoch i, we need to generate the initial secret/public key pair for scheme S_{i+1}. Since key generation for the iterated sum scheme S_{i+1} is proportional to the number of subperiods 2^{i+1}, this can take quite a lot of time. To make the update procedure uniformly fast, we can distribute the computation of this secret/public key pair across the 2^i time periods of the S_i scheme. It is easy to see that the dominant part of the S_{i+1} key generation is the generation of the 2^{i+1} keys associated to the leaves of the tree. If at each update operation of the S_i subtree we generate two leaves for the S_{i+1} scheme, then by the time the ith epoch is over, the key for the new epoch will be ready. So, the total cost of an update operation is given by $3k^2l$ plus a logarithmic number of hashing and pseudorandom generator applications.

This brief analysis shows that the MMM scheme is competitive with all other existing schemes with respect to all parameters and has the added advantage of a practically unbounded number of time periods. Moreover, MMM outperforms each of the previously proposed schemes in at least one parameter. For example, when compared to the scheme of [9], we see that our scheme has much faster key generation and update procedures, while increasing the other parameters only by a small constant factor. Even when [9] is implemented using the "pebbling" technique, our update procedure is superior because it requires $\log t$ hash function computations as opposed to $\log t$ modular exponentiations, and secret key is also much shorter, being only $O(k + l \log t)$ instead of $O(k \log t)$. It should be noted that since our scheme is generic, we can get different performance trade-offs by changing the underlying signature scheme. (Similar trade-offs are possible also for the binary tree scheme, or the scheme of Krawczyk [10], but not for the number theoretic constructions of [3, 2, 9].) For example, if the Rabin signature scheme is used, the verification time drops to $O(k^2)$, outperforming all non-generic schemes. However this makes signature generation slightly slower, going from $O(k^2l)$ to $O(k^3)$. Similarly, signature size can be reduced by choosing an underlying signature scheme with short signatures, possibly at the expense of signing or verification time.

6 Security Analysis

To analyze the security of forward-secure schemes constructed with our composition operations, we associate to each scheme an *insecurity function* $\mathbf{InSec}^{\mathrm{fs}}$, whose value is the maximum probability that an adversary restricted to using specified resources (given as arguments to the function) can break the scheme. In this section, we prove theorem statements relating the insecurity of the new scheme to the insecurity of the underlying schemes.

The insecurity of known forward-secure signature schemes, which we use as building blocks, grows linearly with the number of time periods in the scheme. We briefly explain the intuition for this fact here. To prove the security of such a scheme, we construct an algorithm which, if given a forger for the scheme, can accomplish some task which is believed to be hard (say, factoring a large product of two primes, or forging a signature in a related non-forward-secure scheme). In order for the algorithm to translate the forger's ability into a solution for the problem in question, the forger must produce a forgery for a particular time period. Intuitively, then, this factor of T loss in security arises when the algorithm must guess the particular time period when the forger will produce a valid forgery.

In light of the fact that the number of time periods T in a forward-secure scheme factors into the security of the scheme, we consider the product $\mathbf{InSec}^{\mathrm{fs}} \cdot T$ as a measure of the insecurity of the scheme. Considering this measure, the theorems below demonstrate that both of our composition methods are security-preserving. That is, using either the sum or product composition to join two schemes will result in a scheme whose security is (approximately) the same as the individual schemes' security, relative to the resulting number of time periods.

In the case of the schemes described in Section 5, where the number of time periods is not fixed in advance, the insecurity will grow linearly not with T, the total number of periods in the scheme, but rather with t, the number of periods in the scheme *thus far*, or equivalently the number of update requests made by the adversary. (Since the adversary is constrained to be polynomial time, then this number is always bounded by a polynomial, and the new scheme is guaranteed to satisfy asymptotic security.)

6.1 Security of the Sum Composition

In this section, we state a claim about the security of the sum composition. We assume only that the underlying signature schemes are forward-secure, and that the hash function Hash used in the construction is collision-resistant. In reality, the weaker assumption of a target-collision-resistant hash function would result in a secure construction, but we make the stronger assumption here to simplify the discussion. The proof of the following theorem can be found in the appendix.

Theorem 3 *Let Σ_0 be a key-evolving signature scheme with T_0 time periods, forward-secure against any adversary running in time t_0 and making at most q_0 signature queries. Similarly, let Σ_1 be a key-evolving signature scheme with T_1 time periods, forward-secure against any adversary running in time t_1 and making at most q_1 signature queries. Assume* Hash *is a collision-resistant hash function. Then $\Sigma_0 \oplus \Sigma_1$, the sum composition of Σ_0 and Σ_1, is a new forward-secure scheme such that for any running time t, and number of signature queries up to q:*

$$\mathbf{InSec}^{\mathrm{fs}}(\Sigma_0 \oplus \Sigma_1, t, q) \quad \leq \quad \mathbf{InSec}^{\mathrm{fs}}(\Sigma_0, t_0, q_0) + \mathbf{InSec}^{\mathrm{fs}}(\Sigma_1, t_1, q_1)$$

where both of the following hold:

$$\begin{aligned} t &= \max\{(t_0 - q \cdot \mathrm{SG}_1 - \mathrm{T}_1 \cdot \mathrm{UP}_1 - \mathrm{KG}_1), \\ &\qquad (t_1 - q \cdot \mathrm{SG}_0 - \mathrm{T}_0 \cdot \mathrm{UP}_0 - \mathrm{KG}_0)\} \\ q &\leq \max\{q_0, q_1\} \end{aligned}$$

6.2 Security of the Product Composition

This section gives a security claim about the product composition, assuming only that the underlying signature schemes themselves are forward-secure. Its proof can be found in the appendix.

Theorem 4 *Let Σ_0 be a forward-secure signature scheme with T_0 time periods, and let Σ_1 be a forward-secure signature scheme with T_1 time periods. Then $\Sigma_0 \otimes \Sigma_1$, the product composition of Σ_0 and Σ_1, is a new forward-secure scheme such that for any running time t, and number of signature queries up to q:*

$$\mathbf{InSec}^{\mathrm{fs}}(\Sigma_0 \otimes \Sigma_1, t, q) \quad \leq \quad \mathbf{InSec}^{\mathrm{fs}}(\Sigma_0, t_0, q_0) + \mathrm{T}_0 \cdot \mathbf{InSec}^{\mathrm{fs}}(\Sigma_1, t_1, q_1)$$

where both of the following hold:

$$\begin{aligned} t &\leq \max\{(t_0 - q \cdot \mathrm{SG}_1 - \mathrm{T}_0 \cdot (\mathrm{KG}_1 + \mathrm{T}_1 \cdot \mathrm{UP}_1)), \\ &\quad (t_1 - q \cdot \mathrm{SG}_1 - \mathrm{T}_0 \cdot (\mathrm{KG}_1 + \mathrm{SG}_0 + \mathrm{UP}_0 + \mathrm{T}_1 \cdot \mathrm{UP}_1))\} \\ q &\leq \max\{\mathrm{T}_0, q_1\} \end{aligned}$$

References

[1] R. Anderson. *Two remarks on public-key cryptology.* Manuscript, Sep. 2000. Relevant material presented by the author in an invited lecture at the Fourth ACM Conference on Computer and Communications Security (Apr. 1997).

[2] M. Abdalla and L. Reyzin. *A new forward-secure digital signature scheme.* In Advances in Cryptology - Asiacrypt 2000, LNCS **1976** (Dec. 2000), pp. 116-129.

[3] M. Bellare and S. Miner. *A forward-secure digital signature scheme.* In Advances in Cryptology - CRYPTO '99, LNCS **1666** (Aug. 1999), pp. 431-448.

[4] M. Blum and S. Micali. *How to generate cryptographically strong sequences of pseudorandom bits.* SIAM Journal of Computing **13(4)**(Nov. 1984), pp. 850-864.

[5] S. Even, O. Goldreich, and S. Micali. *On-line/off-line digital signatures.* Journal of Cryptology, vol 9, 1996, pp. 35-67.

[6] O. Goldreich, S. Goldwasser, and S. Micali. *How to construct random functions.* Journal of the ACM, **33(4)**(Oct. 1986), pp. 281–308. Preliminary version in the Proceedings of the IEEE Symposium on the Foundations of Computer Science, 1984, pp 464–479.

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

[8] L.C. Guillou and J.J. Quisquater. *A "paradoxical" identity-based signature scheme resulting from zero-knowledge.* In Advances in Cryptology - CRYPTO '88, LNCS **403** (Aug. 1988), pp. 216-231.

[9] G. Itkis and L. Reyzin. *Forward-secure signatures with optimal signing and verifying.* To appear in Advances in Cryptology - CRYPTO '01, (Aug. 2001).

[10] H. Krawczyk. *Simple forward-secure signatures from any signature scheme.* In Seventh ACM Conference on Computer and Communications Security (Nov. 2000), pp. 108-115.

[11] R. C. Merkle. *A digital signature based on a conventional encryption function.* In Advances in Cryptology - CRYPTO '89, pp. 428- 446.

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

[13] M. Naor and M. Yung. *Universal one-way hash functions and their cryptographic applications.* In Proceedings of the ACM Symposium on Theory of Computing, 1989, pp. 33–43.

[14] M. Rabin. *Digital signatures and public key functions as intractable as factorization.* MIT Laboratory for Computer Science Report TR-212, January 1979.

[15] J. Rompel. *One-way functions are necessary and sufficient for secure signatures.* In Proceedings of the ACM Symposium on Theory of Computing, 1990, pp. 387–394.

[16] A. Yao. *Theory and applications of trapdoor functions.* In Proceedings of the IEEE Symposium on the Foundations of Computer Science, 1982, pp. 80–91.

A Proofs of Security of the Composition Operations

Proof of Theorem 3 (Sketch). We prove the claim in Theorem 3 by assuming the existence of a forger $F_\oplus$, which can break scheme $\Sigma_0 \oplus \Sigma_1$. We then use this forger as a subroutine in two additional forger algorithms, F_0 and F_1, constructed to break schemes Σ_0 and Σ_1, respectively. Each algorithm F_i is given the public key pk_i and a random string r as input, as well as access to a signing oracle $\mathcal{O}_i$ for Σ_i. Both algorithms are described in detail below. Here, we briefly sketch the intuition behind the success probability of algorithms F_0 and F_1.

At some point during its execution, $F_\oplus$ will enter its break-in phase, and request to learn the secret key for $\Sigma_0 \oplus \Sigma_1$. Let $e \in \{0,1\}$ denote whether this occurred during the first or second portion of the scheme, that is, during its Σ_0 or the Σ_1 portion. Assume that $F_\oplus$ generates a successful forgery, denoted by $(M, (t, \langle \sigma, pk_0, pk_1 \rangle))$. Define $e' = 0$ if forgery time period $t < \mathrm{T}_0$ and $e' = 1$ otherwise (that is, let e' denote the underlying scheme for which $F_\oplus$ produces a forgery, either Σ_1 or Σ_2). Let $\mathcal{E}_1$ denote the event where $e == 0$. Let $\mathcal{E}_2$ denote the event that $e' == e$, that is, that the forgery attempt is during the same epoch as the break-in. We now consider several cases.

- Case I: Event $\mathcal{E}_1$ occurs. Since the forgery was successful, and the break-in occurred during Σ_0, then the forgery was for a time period during Σ_0 before the break-in. This indicates that our F_0 will win.

- Case II: Event $\mathcal{E}_1$ does not occur, and $\mathcal{E}_2$ does not occur. This means that the break-in occurred during Σ_1, but the forgery time was during Σ_0. Since the knowledge learned by the forger $F_\oplus$ during the break-in is unrelated to the scheme for which it forged, this case means that $F_\oplus$ must have found a weakness in the Σ_0 scheme (and didn't even use a key from a later time period within Σ_0 to construct the forgery). Exploiting this, our F_0 will win.

- Case III: Event $\mathcal{E}_1$ does not occur, but $\mathcal{E}_2$ does. This means that the adversary broke in during Σ_1, and created a forgery during that same portion of the scheme. In this case, our forger F_1 will succeed.

This allows us to write:

$$
\begin{aligned}
\Pr[\,F_0\,] &\geq \Pr[\,F_\oplus \cap \mathcal{E}_1\,] + \Pr[\,F_\oplus \cap \overline{\mathcal{E}_1} \cap \overline{\mathcal{E}_2}\,], \text{ and} \\
\Pr[\,F_1\,] &\geq \Pr[\,F_\oplus \cap \overline{\mathcal{E}_1} \cap \mathcal{E}_2\,].
\end{aligned}
$$

The proof of Theorem 3 follows from the following bound on the probability that $F_\oplus$ succeeds:

$$
\begin{aligned}
\Pr[\,F_\oplus\,] &= \Pr[\,F_\oplus \cap \mathcal{E}_1\,] + \Pr[\,F_\oplus \cap \overline{\mathcal{E}_1}\,] \\
&= \Pr[\,F_\oplus \cap \mathcal{E}_1\,] + \Pr[\,F_\oplus \cap \overline{\mathcal{E}_1} \cap \overline{\mathcal{E}_2}\,] + \Pr[\,F_\oplus \cap \overline{\mathcal{E}_1} \cap \mathcal{E}_2\,] \\
&\leq \Pr[\,F_0\,] + \Pr[\,F_1\,].
\end{aligned}
$$

Algorithms for the Proof of Security of the Sum Construction.

Algorithm $F_0^{\mathcal{O}_0}(pk_0, r)$

$q' \leftarrow 0$; $e \leftarrow 0$; $p \leftarrow 0$; $(r_0, r_1) \leftarrow G(r)$; $(sk_{1,0}, pk_1) \leftarrow KeyGen_1(r_1)$; Delete $sk_{1,0}$
Simulate the environment of $F_\oplus$ as follows: $pk_\oplus \leftarrow \mathrm{Hash}(pk_0, pk_1)$; Give $pk_\oplus$ to $F_\oplus$
While $F_\oplus$ is still requesting action
If (request == update time period) then
If ($e == 0$) then
If ($p == \mathrm{T}_0 - 1$)then Delete $sk_{0,p}$; $e \leftarrow 1$; $p \leftarrow 0$; $(sk_{1,p}, pk_1) \leftarrow KeyGen_1(r_1)$
Else $p \leftarrow p + 1$; F_0 requests to update time period
End if
Else If ($p == \mathrm{T}_1 - 1$) then Output "Update past last period" and HALT; End if
$p \leftarrow p + 1$; Call $Update_1()$
End if
Else if (request == signature query on M) then
$q' \leftarrow q' + 1$
If ($q' > q$) then Output "Too many signature queries" and HALT ; End if
If ($e == 0$) then
F_0 requests signature query on M, receives σ
Else $\sigma \leftarrow Sign_1(sk_{1,p}, M)$
End if
$t \leftarrow e \cdot \mathrm{T}_0 + p$; Return $(\langle \sigma, pk_0, pk_1 \rangle, t)$ to $F_\oplus$
Else if (request == break-in) then
If ($e == 0$) then
F_0 requests to break in and learn $sk_{0,p}$; Return $(sk_{0,p}, r_1, pk_0, pk_1)$ to $F_\oplus$
Else Return $(sk_{1,p}, \epsilon, pk_0, pk_1)$ to $F_\oplus$
End if
Else Output "invalid request" and HALT; End if
End while
$(M, (\langle \sigma, pk_0, pk_1 \rangle, t)) \leftarrow F_\oplus$'s attempted forgery
Stop running $F_\oplus$
If ($e == 1$) then Output "Unsuccessful in breaking Σ_0" and HALT; End if
Return $(M, (\sigma, t - T_0))$ as attempted forgery for scheme Σ_0

Algorithm $F_1^{\mathcal{O}_1}(pk_1, r)$

$q' \leftarrow 0$; $e \leftarrow 0$; $p \leftarrow 0$; $(r_0, r_1) \leftarrow G(r)$; $(sk_{0,0}, pk_0) \leftarrow KeyGen_0(r_0)$
Simulate the environment of $F_\oplus$ as follows: $pk_\oplus \leftarrow \mathrm{Hash}(pk_0, pk_1)$; Give $pk_\oplus$ to $F_\oplus$
While $F_\oplus$ is still requesting action
If (request == update time period) then
If ($e == 0$) then
If ($p == \mathrm{T}_0 - 1$)then Delete $sk_{0,p}$; $e \leftarrow 1$; $p \leftarrow 0$
Else $p \leftarrow p + 1$; Call $Update_0()$
End if
Else If ($p == \mathrm{T}_1 - 1$) then Output "Update past last period" and HALT; End if
$p \leftarrow p + 1$; F_1 requests to update time period
End if
Else if (request == signature query on M then
$q' \leftarrow q' + 1$
If ($q' > q$) then Output "Too many signature queries" and HALT; End if
If ($e == 0$) then $\sigma \leftarrow Sign_0(sk_{0,p}, M)$
Else F_1 requests signature query on M, receives σ
End if

$t \leftarrow e \cdot \mathrm{T}_0 + p$; Return $(\langle \sigma, pk_0, pk_1 \rangle, t)$ to $F_\oplus$
Else if (request == break-in) then
If ($e == 0$) then Output "Simulation of $F_\oplus$'s environment failed" and HALT.
Else F_1 requests to break in and learn $sk_{1,p}$; Return $((sk_{1,p}, \epsilon), pk_0, pk_1)$ to $F_\oplus$
End if
Else Output "invalid request" and HALT
End if
End while
$(M, (\langle \sigma, pk_0, pk_1 \rangle, t)) \leftarrow F_\oplus$'s attempted forgery
Stop running $F_\oplus$
If ($e == 0$) then Output "Unsuccessful in breaking Σ_1" and HALT; End if
Return $(M, (\sigma, t - \mathrm{T}_0))$ as attempted forgery for scheme Σ_1.

Proof of Theorem 4 (Sketch). The proof for the product case uses the same technique as the proof for the sum composition. We assume the existence of forger $F_\otimes$, and construct new algorithms F_0 and F_1 to break the underlying schemes. Again, each algorithm F_i is given the public key pk_i and random string r as input, as well as access to a signing oracle $\mathcal{O}_i$ for Σ_i. Both algorithms F_0 and F_1 are described in detail below, but first we sketch the intuition behind their success probability, to justify the claim.

Assume that $F_\otimes$ generates a successful forgery, denoted $(M, (\langle pk, \sigma_0, \sigma_1 \rangle, t))$. In the $\Sigma_0 \otimes \Sigma_1$ scheme, we have multiple instantiations of the scheme Σ_1, one for each time period of the Σ_0 scheme, or "epoch". Because we begin numbering time periods (and hence epochs) at 0, a forgery during scheme time period t is a forgery during epoch $e' = \lfloor \frac{t}{\mathrm{T}_1} \rfloor$. To distinguish public keys of instantiations of Σ_1 in different epochs, we will superscript each pk_1 with its epoch number.

Assume that, in its break-in phase, $F_\otimes$ learned the secret key for $\Sigma_0 \otimes \Sigma_1$ at some time period during epoch e. Now, let $\mathcal{SEEN}$ = the set of public keys for the Σ_1 scheme instantiations which the forger $F_\otimes$ sees during its sign queries, e.g., $\mathcal{SEEN} = \{ {pk_1}^{(0)}, {pk_1}^{(1)}, \ldots, {pk_1}^{(e)} \}$. Let $\mathcal{E}$ denote the event where ${pk_1}^{(e')} \notin \mathcal{SEEN}$. Because the forgery attempt was is successful, it must be the case that all of the following hold:

$$\begin{array}{l} \mathit{Verify}_0({pk_1}^{(e')}, \sigma_0) == 1. \\ \mathit{Verify}_1(M, \sigma_1) == 1. \\ F_\otimes \text{ did not ask } M \text{ as a query during time } t. \end{array}$$

Now, if event $\mathcal{E}$ occurs, then σ_0 represents a successful forgery against the Σ_0 scheme on a new message ${pk_1}^{(e')}$, meaning our forger F_0 will win. On the other hand, if $F_\otimes$ succeeds but event $\mathcal{E}$ does not occur, then σ_1 must be a successful forgery for the Σ_1 scheme on message M. Since M was never queried in time t of the $\Sigma_0 \otimes \Sigma_1$ scheme, then it is clear it was never queried in time $t \bmod \mathrm{T}_0$ of the Σ_1 scheme during epoch e'. So M is a new message. However, this does not necessarily mean that F_1, forger we have constructed to break Σ_1, wins. In order to succeed, F_1 must correctly select an epoch f ahead of time, a guess for in which epoch $F_\otimes$ will forge. F_1 selects f at random from all possible epochs, though, so the probability that F_1 guesses corectly is one in T_0. This reduces the overall success probability of F_1 by a factor of T_0.

So we have:

$$\begin{array}{rcl} \Pr[\, F_0 \,] & \geq & \Pr[\, F_\otimes \cap \mathcal{E} \,], \text{ and} \\ \Pr[\, F_1 \,] & \geq & \dfrac{1}{\mathrm{T}_0} \cdot \Pr[\, F_\otimes \cap \overline{\mathcal{E}} \,]. \end{array}$$

Theorem 4 follows from the following bound on the probability that $F_\otimes$ succeeds:

$$\begin{aligned}\Pr[\,F_\otimes\,] &= \Pr[\,F_\otimes \cap \mathcal{E}\,] + \Pr[\,F_\otimes \cap \overline{\mathcal{E}}\,] \\ &\leq \Pr[\,F_0\,] + \mathrm{T}_0 \cdot \Pr[\,F_1\,].\end{aligned}$$

Algorithms for the proof of security of the product construction.

Algorithm $F_0^{\mathcal{O}_0}(pk_0, r)$
- $q' \leftarrow 0$; $e \leftarrow 0$; $p \leftarrow 0$; $(r'_1, r''_1) \leftarrow G(r)$; $(sk_{1,p}, pk_1{}^{(e)}) \leftarrow KeyGen_1(r'_1)$
- F_0 requests a signature query on message $pk_1{}^{(e)}$: $\sigma_0 \leftarrow \mathcal{O}_\prime(pk_1{}^{(e)})$
- Simulate the environment of $F_\otimes$ as follows:
  - $pk_\otimes \leftarrow pk_0$; Give $pk_\otimes$ to $F_\otimes$
  - While $F_\otimes$ is still requesting action
    - If (request == update time period) then $p \leftarrow p + 1$
      - If $(p == \mathrm{T}_1)$ then
        - $e \leftarrow e + 1$
        - If $(e == \mathrm{T}_0)$ then Output "Update past last period" and HALT; End if
        - F_0 requests to update time period; $p \leftarrow 0$
        - $(r'_1, r''_1) \leftarrow G(r''_1)$; $(sk_{1,p}, pk_1{}^{(e)}) \leftarrow KeyGen_1(r'_1)$
        - F_0 requests a signature query on message $pk_1{}^{(e)}$: $\sigma_0 \leftarrow \mathcal{O}_0(pk_1{}^{(e)})$
      - Else Call $Update_1()$, which computes $sk_{1,p}$
      - End if
    - Else if (request == signature query on M) then
      - $q' \leftarrow q' + 1$
      - If $(q' > q)$ then Output "Too many signature queries" and HALT; End if
      - $\sigma_1 \leftarrow Sign_1(sk_{1,p}, M)$; $t \leftarrow e \cdot \mathrm{T}_1 + p$
      - Return $(t, \langle pk_1{}^{(e)}, \sigma_0, \sigma_1 \rangle)$
    - Else if (request == break-in) then
      - F_0 requests to break-in, learns $sk_{0,e}$
      - Return $(sk_{0,e}, \sigma_0, sk_{1,p}, pk_1{}^{e}, r'_1)$ to $F_\otimes$ as the secret key for $\Sigma_0 \otimes \Sigma_1$
    - Else Output "Invalid request" and HALT
    - End if
  - End while
  - $(M, (\langle pk_1{}^{(e')}, \sigma_0, \sigma_1 \rangle, t)) \leftarrow F_\otimes$'s attempted forgery
- Stop running $F_\otimes$
- Return $(pk_1{}^{(e')}, \sigma_0)$ as attempted forgery for scheme F_0

Algorithm $F_1^{\mathcal{O}_1}(pk_1, r)$
- $q' \leftarrow 0$; $f \xleftarrow{R} \{0, 1, \ldots, \mathrm{T}_0 - 1\}$; $pk_1{}^{(f)} \leftarrow pk_1$; $(r_0, r_1) \leftarrow G(r)$; $(r'_1, r''_1) \leftarrow G(r_1)$
- $(sk_{0,0}, pk_0) \leftarrow KeyGen_0(r_0)$; $e \leftarrow 0$; $p \leftarrow 0$
- If $(e \neq f)$ then $(sk_{1,p}, pk_1{}^{(e)}) \leftarrow KeyGen_1(r'_1)$; End if
- $\sigma_0 \leftarrow Sign_0(sk_{0,e}, pk_1{}^{(e)})$; Call $Update_0()$
- Simulate the environment of $F_\otimes$ as follows:
  - $pk_\otimes \leftarrow pk_0$; Give $pk_\otimes$ to $F_\otimes$
  - While $F_\otimes$ is still requesting action
    - If (request == update time period) then
      - $p \leftarrow p + 1$
      - If $(p == \mathrm{T}_1)$ then
        - $e \leftarrow e + 1$
        - If $(e == \mathrm{T}_0)$ then Output "Update past last period" and HALT; End if
        - $p \leftarrow 0$

If $(e \neq f)$ then $(r_1', r_1'') \leftarrow G(r_1'')$; $(sk_{1,p}, pk_{1,e}) \leftarrow KeyGen_1(r_1')$; End if
$\sigma_0 \leftarrow Sign_0(sk_{0,e}, {pk_1}^{(e)})$; Call $Update_0()$
Else
If $(e == f)$ then F_1 requests to update time period
Else Call $Update_1()$
End if
End if
Else if (request == signature query on M) then
$q' \leftarrow q' + 1$
If $(q' > q)$ then Output "Too many signature queries" and HALT; End if
If $(e == f)$ then F_1 requests a signature on message M: $\sigma_1 \leftarrow \mathcal{O}_1(M)$
Else $\sigma_1 \leftarrow Sign_1(sk_{1,e}, M)$
End if
$t \leftarrow e \cdot \mathrm{T}_1 + p$; Return $(\langle {pk_1}^{(e)}, \sigma_0, \sigma_1 \rangle, t)$
Else if (request == break-in) then
If $(e == f)$ then F_1 requests break-in, learns $sk_{1,p}$ for epoch e; End if
Return $((sk_{0,e}, \sigma_0, sk_{1,p}), {pk_1}^{e}, r_1')$ to $F_\otimes$ as secret key for $\Sigma_0 \otimes \Sigma_1$
Else Output "Invalid request" and HALT; End if
End while
$(M, (\langle {pk_1}^{(e')}, \sigma_0, \sigma_1 \rangle), j) \leftarrow F_\otimes$'s attempted forgery
Stop running $F_\otimes$
$e' \leftarrow \lfloor \frac{t}{\mathrm{T}_1} \rfloor$; $p' \leftarrow t - \mathrm{T}_1 \cdot e'$
If $(e' \neq f)$ then Output "Incorrect guess for forgery epoch" and HALT; End if
Return (M, σ_1) as attempted forgery for Σ_1 scheme