Efficient Generic Forward-Secure Signatures With An Unbounded Number Of Time Periods

Tal Malkin[1], Daniele Micciancio[2,⋆], and Sara Miner[2,⋆⋆]

[1] AT&T Labs Research
180 Park Avenue, Florham Park, NJ 07932, USA.
E-mail: tal@research.att.com.
[2] University of California, San Diego
Department of Computer Science and Engineering, Mail Code 0114
9500 Gilman Drive, La Jolla, CA 92093-0114, USA.
E-mail: {daniele, sminer}@cs.ucsd.edu.

Abstract. We construct the first efficient forward-secure digital 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 number of time periods for which our scheme can be used is bounded only by an exponential function of the security parameter (given this much time, any scheme can be broken by exhaustive search), and its 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. As an intermediate step in designing our scheme, we propose and study two general composition operations that can be used to combine *any* existing signature schemes (whether standard or forward-secure) into new forward-secure signature schemes.

1 Introduction

The standard notion of digital signature security is extremely vulnerable to leakage of the secret key, which, over the lifetime of the scheme, may be quite a realistic threat. Indeed, if the secret key is compromised, any message can be forged. All future signatures are invalidated as a result of such a compromise, and, furthermore, no *previously issued* signatures can 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

⋆ Supported in part by NSF Career Award CCR-0093029
⋆⋆ Supported in part by a Graduate Diversity Fellowship from the San Diego Supercomputer Center, and Mihir Bellare's 1996 Packard Foundation Fellowship in Science and Engineering

trusting the signer. For example, it is very easy for a dishonest signer to leak his secret key in order to repudiate a previously signed document. Furthermore, changing the schemes keys very frequently is also not a practical solution, since frequently 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 [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 revoked without invalidating signatures issued during earlier time periods.

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 and time. Typically, these schemes achieve efficiency in some parameters at the price of making other parameters significantly worse than in standard signature schemes. For example, some schemes have faster signature generation and verification algorithms, but slower key generation and update procedures, while other schemes have the opposite behavior. In other cases, time improvements are obtained at the cost of larger signatures or larger secret and public keys. Moreover, in essentially all previous schemes, although the number of time periods can be arbitrarily large, its value must be set in advance, and passed as a parameter during the key generation process. The performance of the algorithms then depends on the security parameter as well as the a priori maximum number of time periods T. So, setting T to an unnecessarily large number results in a considerable efficiency loss.

Clearly, there is a trade-off between the efficiency parameters, and which scheme is most practical 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 signatures themselves shorter. On the other hand, consider an electronic checkbook (e-check) application where the time period corresponds to the check serial number, rather than physical time. If your electronic wallet is stolen (with the current e-check secret key in it), you want 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 design a new tree-like digital signature scheme which we call the MMM scheme. Our construction uses the idea of Merkle trees [10], which were suggested for use in the forward security context by Krawczyk [9]. Our scheme is a *generic* construction, namely it can be based on any underlying signature scheme, and does not rely on specific computational assumptions like discrete log or factoring. Moreover, the security of MMM can be proved in the standard complexity model, without resorting to the random oracle methodology. This scheme is very efficient in all parameters (simultaneously), and outperforms all previous constructions in at least some parameters (the price is never more than a constant increase in other parameters, and in most cases there is no price at all). In fact, our scheme is asymptotically essentially optimal, while in practice it is competitive even with standard signature schemes. For example, signing requires a single signature in the underlying scheme, and verifying requires little more than two verifications of the underlying scheme. (More performance details are given later.) Furthermore, the MMM scheme is the first efficient forward-secure scheme which does not require its maximal number of time periods 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 cannot 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). To summarize, the significance of our scheme stems from the following properties:

- **Practical:** MMM has excellent performance for practical parameters. This is further improved by the fact that the maximal number of time periods is not a relevant parameter. Any standard signature scheme can be used as a building block, providing trade-offs between the parameters, thus allowing greater flexibility in accommodating the requirements of particular applications, and meeting practical constraints.
- **Strong security guarantee:** The scheme is secure under the minimal necessary assumption, namely that (standard) digital signature schemes exist (which is equivalent to the existence of one-way functions [15]). This security is proved without relying on the random oracle model.

As an intermediate step in developing the MMM scheme, we propose and study two general composition operations that can be used to combine *any* existing signature schemes (whether standard or forward-secure) into new forward-secure schemes. We prove their security in the standard complexity model, based on the security of the underlying schemes, and without requiring any additional assumption.[1] To obtain our main result, we then extend these composition operations to generate the MMM construction. Separately, in Section 5, we present several additional constructions (some new and some already known) derived via these composition operations, which illustrate some of the efficiency tradeoffs possible.

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

1.2 Related Work

Recently, 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 found 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. We use two different security parameters, as follows:

- 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 such that more generally common number theoretic problems (such as inverting the RSA function) become infeasible.

It is important to distinguish between the two values because private key cryptography is typically much more efficient than public key. 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, $k = 1000$ and $l = 100$ are acceptable values. In the analysis of all constructions we assume that modular multiplications of k-bit numbers are performed in k^2 time[2] 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 a modular arithmetic operation.

In the tables below, we omit small constant factors and 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 $l = o(k)$.

[2] Although faster multiplication algorithms are known, we feel that our assumptions accurately represent the relationship between the speeds of private and public key applications in practice.

The previous schemes we analyze below 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.

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 [9]. Their performance is summarized in Figure 1. As usual, T stands for the total 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 [7], and of Micali [11]. 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	$lk^2 \log T$	lk^2T
Signing time	lk^2	lk^2	lk^2
Verification time	lk^2	$lk^2 \log T$	lk^2
Key update time (*)	$O(1)$	lk^2	lk^2
Secret key size	kT	$k \log T$	k
Non-secret storage	kT	$k \log T$	kT
Public key size	k	k	k
Signature size	k	$k \log T$	k

Fig. 1. Comparing parameters of generic constructions [1, 3, 9]. (*)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 of $\log T$.

As shown in Figure 1, 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, 9].

The above constructions are generic, meaning they may be instantiated with any signature scheme. Their proofs of security rely only on the security of the underlying signature scheme (in particular, none of these security proofs rely on random oracles).[3]

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 [8]. All these are proven to be secure *in the random oracle model*, assuming that factoring is hard (for [3, 2]) or that taking any root modulo a composite is hard (for [8]). The performance of these schemes is summarized in Figure 2.

	BM	AR	IR	IR'
Key gen time	lk^2T	lk^2T	$k^5+(k+l^3)lT$	$k^5+T\log^4 T+k^2l+klT$
Signing time	$(T+l)k^2$	lk^2T	k^2l	k^2l
Verification time	$(T+l)k^2$	lk^2T	k^2l	k^2l
Key update time	lk^2	lk^2	$(k^2+l^3)lT$	$(k^2l+l^2+\log^4 T)\log T$
Secret key size	lk	k	k	$(1+\log T)k$
Public key size	lk	k	k	k
Signature size	k	k	k	k

Fig. 2. Comparing parameters of schemes built on specific security assumptions [3, 2, 8]. IR' denotes the Itkis-Reyzin variant using the two optimizations from [8].

Except for the key generation and 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 [8] also suggest a variant of their scheme where the update time is only proportional to $\log T$, but the secret key size increases by a factor $\log T$, thus matching generic constructions like the binary tree scheme. (In fact, generic constructions 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 $\log T = o(l)$. (In the full version of this paper [12], we show a different tree construction, in which even faster verification times are possible.)

[3] Of course, when instantiated with a specific base signature scheme which requires other assumptions, such as a random oracle, the resulting forward-secure scheme also requires the same assumptions.

Comparison to Our Results. All previous known schemes (with the exception of the very inefficient "long signature scheme" described in [3]) require the total (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 $\log T$, but in most cases linearly for some parameters). We propose a new scheme, MMM, whose performance avoids any dependence on T, and depends only on the number of time periods elapsed so far (even this dependence is minimal). In fact, T need never be fixed. In addition, MMM also combines the best of both types of previous constructions in terms of efficiency and security. Indeed, we construct it 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.

In the left half of Figure 3, we demonstrate the efficiency of the scheme when instantiated with the Guillou-Quisquater signature scheme.[4] In the right half of that same figure, we select specific parameter values, and show an actual efficiency comparison between our scheme and the best previous schemes in each category, namely the two Itkis-Reyzin schemes, and the Bellare-Miner binary tree scheme. Relative to IR, note that we achieve the same or better improvements over the binary tree scheme, without paying the price that IR pays in other parameters. Also note that while we instantiated $t = 1000$ in MMM (the same value as the instantiated *maximal* T in the other schemes), most of the time t will be much smaller, implying better performance. More details about the performance of our scheme can be found in Section 4.

2 Definitions

As formalized in [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

[4] In order to compare a generic construction with a construction using specific security assumptions, we need to select a specific base scheme for the generic construction. Here, GQ is a good selection for comparison, as it is efficient, and it is the one underlying the Itkis-Reyzing constructions as well. Because the GQ scheme is proven secure in the random oracle model, the proof of forward security of the resulting scheme also requires random oracles. This particular example does not change the fact that our security proof does not need them itself.

	MMM
Key gen time	$k^2 l^2$
Signing time	$k^2 l$
Verification time	$k^2 l + l^2 \log l$
Key update time (*)	$k^2 l + (k + l^2) \log t$
Secret key size	$k + l \log l$
Public key size	l
Signature size	$k + l \log l$

	MMM	IR	IR'	B.T.
Key gen time	10^{10}	10^{15}	10^{15}	10^9
Signing time	10^8	10^8	10^8	10^8
Verification time	10^8	10^8	10^8	10^9
Key update time	10^8	10^{11}	10^9	10^8
Secret key size	10^3	10^3	10^4	10^4
Public key size	10^2	10^3	10^3	10^3
Signature size	10^3	10^3	10^3	10^4

Fig. 3. Analyzing the MMM scheme instantiated with GQ. We first present an asymptotic analysis. Then, assuming parameter values $k = 1000$, $l = 100$, and $T = 1000$, we compare estimated times and sizes for MMM with the IR schemes and the Bellare-Miner binary tree scheme. In the case of MMM, we provide worst-case values, by setting $t = 1000$. (∗) The running time of the update algorithm is amortized. The worst case update may take tlk^2. (Later, we provide another version of our scheme which achieves uniformly fast update time, at the price of additional $l \log^2 t + k \log l$ in the secret key size.)

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 the secret key was leaked. Bellare and Miner formalized the adversary model for forward-secure signature schemes, and the experiment used to define the *insecurity function* for a scheme. The reader is referred to [3] for details.

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, most update, sign and verify algorithms need t (and perhaps even T) to work properly, and therefore these values should be thought as integral parts of keys and signatures. Below, we summarize the general input and output parameters of each algorithm of a forward-secure digital signature scheme. We note that for the MMM scheme in particular, an input T is not needed for any of its algorithms, and thus can be omitted below.

– 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 M, and outputs a signature (σ, t) on M.
– 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.

3 Composition Methods

In this section, we describe two general composition methods that can be used to combine both standard and forward-secure signature schemes into forward-secure schemes with more time periods. These are methods which have been used implicitly in previous constructions (e.g., in [10, 3], see Sect. 5), and we formalize them here. Although these compositions are interesting in their own right, we present them here as a means to lead into our main construction, which can be found in Section 4. To illustrate the flexibility afforded by these composition operations, however, we present additional constructions in Section 5.

In order to unify the presentation, we regard standard signature schemes as forward-secure signature schemes with one time period, namely $T=1$. Let Σ_0 and Σ_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 operations the "sum" and the "product" compositions, respectively. In the following two subsections, we give descriptions of the operations, and give theorems analyzing their security and performance.

Note that 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. These specialized versions are called $\mathrm{SKeyGen}_i$ and $\mathrm{PKeyGen}_i$ respectively. As we shall see, the diversification of KeyGen to PKeyGen and SKeyGen plays a critical role for the performance of forward-secure signature schemes obtained by the iterated application of the basic composition operations.

3.1 The "Sum" Composition

Given any two schemes Σ_0 and Σ_1 as described above, 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 4. We denote the composition of the sum algorithm with itself iteratively to achieve T time periods, by $S^{\oplus}_{\log T}$ (since it looks like a tree with $\log T$ levels). This scheme will later be used (with varying T's) to construct the MMM scheme (and it is further discussed in Section 5).

Algorithm KeyGen(r):
    $(r_0, r_1) \leftarrow G(r)$
    $(sk_0, pk_0) \leftarrow \mathrm{KeyGen}_0(r_0)$
    $pk_1 \leftarrow \mathrm{PKeyGen}_1(r_1)$
    $pk \leftarrow \mathrm{Hash}(pk_0, pk_1)$
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 $sk' \leftarrow \mathrm{Update}_0(t, sk')$
	Else If $(t+1 = T_0)$
	    Then $sk' \leftarrow \mathrm{SKeyGen}_1(r_1)$; $r_1 \leftarrow 0$
	    Else $sk' \leftarrow \mathrm{Update}_1(t - T_0, sk')$
	End

Algorithm Sign$(t, \overbrace{\langle sk', r_1, pk_0, pk_1 \rangle}^{sk}, M)$
If $(t < T_0)$
Then $\sigma' \leftarrow \mathrm{Sign}_0(t, sk', M)$
Else $\sigma' \leftarrow \mathrm{Sign}_1(t - T_0, sk', M)$
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 $\mathrm{Verify}_0(pk_0, M, \sigma, t)$
	    Else $\mathrm{Verify}_1(pk_1, M, \sigma, t - T_0))$
	Else Reject

Fig. 4. The algorithms defining the sum composition.

Security Analysis. Here 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 full version of this paper [12].

Theorem 1. *Let Σ_0 be a key-evolving signature scheme with* T_0 *time periods, forward-secure against any adversary running in time m_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 m_1 and making at most q_1 signature queries. 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 Σ_i for $i = 0, 1$. 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 m, and number of signature queries up to q:*

$$\mathbf{InSec}^{\mathrm{fs}}(\Sigma_0 \oplus \Sigma_1, m, q) \leq \mathbf{InSec}^{\mathrm{fs}}(\Sigma_0, m_0, q_0) + \mathbf{InSec}^{\mathrm{fs}}(\Sigma_1, m_1, q_1)$$

where both of the following hold:

$$\begin{aligned} m &= \max\{(m_0 - q \cdot \mathrm{SG}_1 - \mathrm{T}_1 \cdot \mathrm{UP}_1 - \mathrm{KG}_1), (m_1 - q \cdot \mathrm{SG}_0 - \mathrm{T}_0 \cdot \mathrm{UP}_0 - \mathrm{KG}_0)\} \\ q &\leq \max\{q_0, q_1\} \end{aligned}$$

The theorem above demonstrates that the sum composition is security-preserving. That is, the sum of two schemes will have roughly the same security as the individual schemes' security, relative to the resulting number of time periods.

Performance Analysis. We now give an analysis of key and signature sizes, as well as running times for the sum composition. The relations below can be verified by inspection of Figure 4.

Theorem 2. *Let* $\mathrm{SK_i}$, $\mathrm{PK_i}$ *and* $\mathrm{SIG_i}$ *be the secret key, public key and signature sizes for the forward-secure signature scheme Σ_i for $i = 0, 1$. Then, the key and signature sizes of the sum scheme $\Sigma_0 \oplus \Sigma_1$ are:*

$$\begin{aligned} \mathrm{PK} &= l \\ \mathrm{SK} &= \max(\mathrm{SK}_0, \mathrm{SK}_1) + \mathrm{PK}_0 + \mathrm{PK}_1 + l \\ \mathrm{SIG} &= \max(\mathrm{SIG}_0, \mathrm{SIG}_1) + \mathrm{PK}_0 + \mathrm{PK}_1 \end{aligned}$$

Theorem 3. *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 the forward-secure signature scheme*

Σ_i *for* $i = 0, 1$. *Then the running times of the sum scheme* $\Sigma_0 \oplus \Sigma_1$ *are:* [5]

$$\begin{aligned}
\mathrm{KG} &= \mathrm{KG}_0 + \mathrm{KG}_1 + l^2 \\
\mathrm{SG} &= \max\{\mathrm{SG}_0, \mathrm{SG}_1\} \\
\mathrm{VF} &= \max\{\mathrm{VF}_0, \mathrm{VF}_1\} + l^2 \\
\mathrm{UP} &= (\mathrm{SKGtime}_1 + (T_0 - 1)\mathrm{UP}_0 + (T_1 - 1)\mathrm{UP}_1)/(T_0 + T_1 - 1)
\end{aligned}$$

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 small. In this case, the update algorithm will be more complex, as partial computation of a future update will be performed at each stage. We discuss this below.

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. When composing the sum algorithm with itself to achieve T time periods, the space required to generate the initial secret key for the right child is logarithmic. This means that, after iterating, the total secret key size will be the size of the original secret key (say k), plus $O(l \log^2 T)$. The other parameters of $S^{\oplus}_{\log T}$ (see full version [12]) do not change, and the update time becomes efficient even in the worst case.

3.2 The "Product" Composition

Beginning with any Σ_0 and Σ_1 as described in the beginning of Section 3, we now 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 as 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 5.

[5] As explained in Section 1.2, we assume that a call to a hash function or PRG takes time $O(l^2)$. Also, the update time here is the result of amortized analysis; it represents the average case.

Algorithm KeyGen(r) $(r_0, r_1) \leftarrow G(r)$ $(r'_1, r''_1) \leftarrow G(r_1)$ $(sk_0, pk) \leftarrow \mathrm{KeyGen}_0(r_0)$ $(sk_1, pk_1) \leftarrow \mathrm{KeyGen}_1(r'_1)$ $\sigma \leftarrow \mathrm{Sign}_0(0, sk_0, pk_1)$ $sk_0 \leftarrow \mathrm{Update}_0(0, sk_0)$ Return $(\underbrace{\langle sk_0, \sigma, sk_1, pk_1, r''_1 \rangle}_{sk}, pk)$	Algorithm Update$(t, \overbrace{\langle sk_0, \sigma, sk_1, pk_1, r \rangle}^{sk})$ If $(t + 1 \neq 0 \bmod T_1)$ Then $\mathrm{Update}_1(t \bmod T_1, sk_1)$ Else $(r', r) \leftarrow G(r)$ $(sk_1, pk_1) \leftarrow \mathrm{KeyGen}_1(r')$ $\sigma \leftarrow \mathrm{Sign}_0(\lfloor \frac{t}{T_1} \rfloor, sk_0, pk_1)$ $sk_0 \leftarrow \mathrm{Update}_0(\lfloor \frac{t}{T_1} \rfloor, sk_0)$ End
Algorithm Sign$(t, \overbrace{\langle sk_0, \sigma_0, sk_1, pk_1, r \rangle}^{sk}, M)$ $\sigma_1 \leftarrow \mathrm{Sign}_1(sk_1, M, t \bmod T_1)$ Return $(\underbrace{\langle pk_1, \sigma_0, \sigma_1 \rangle}_{\sigma}, t)$	Algorithm Verify$(pk, M, \overbrace{\langle pk_1, \sigma, \sigma' \rangle}^{\sigma}, t)$ $v_0 \leftarrow \mathrm{Verify}_0(pk, pk_1, \sigma, \lfloor \frac{t}{T_1} \rfloor)$ $v_1 \leftarrow \mathrm{Verify}_1(pk_1, M, \sigma', t \bmod T_1)$ Return $(v_0 \wedge v_1)$

Fig. 5. The algorithms defining the product composition.

Security Analysis. Next, we give 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 full version of this paper [12].

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. 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 Σ_i for $i = 0, 1$. 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 m, and number of signature queries up to q:*

$$\mathbf{InSec}^{\mathrm{fs}}(\Sigma_0 \otimes \Sigma_1, m, q) \leq \mathbf{InSec}^{\mathrm{fs}}(\Sigma_0, m_0, q_0) + \mathrm{T}_0 \cdot \mathbf{InSec}^{\mathrm{fs}}(\Sigma_1, m_1, q_1)$$

where both of the following hold:

$$\begin{aligned} m &\leq \max \{(m_0 - q \cdot \mathrm{SG}_1 - \mathrm{T}_0 \cdot (\mathrm{KG}_1 + \mathrm{T}_1 \cdot \mathrm{UP}_1)), \\ &\qquad (m_1 - q \cdot \mathrm{SG}_1 - \mathrm{T}_0 \cdot (\mathrm{KG}_1 + \mathrm{SG}_0 + \mathrm{UP}_0 + \mathrm{T}_1 \cdot \mathrm{UP}_1))\} \\ q &\leq \max \{\mathrm{T}_0, q_1\} \end{aligned}$$

The above theorem demonstrates that a scheme generated using the product composition will have roughly the same insecurity as the underlying two schemes, relative to the resulting number of time periods. That is, the product construction is security-preserving.

Performance Analysis. For the product construction, we now give an analysis of key and signature sizes, as well as running times. The relations can be verified by inspection of Figure 5.

Theorem 5. *Let* $\mathrm{SK_i}$*,* $\mathrm{PK_i}$ *and* $\mathrm{SIG_i}$ *be the secret key, public key and signature sizes for the forward-secure signature scheme* Σ_i *for* $i = 0, 1$*. Then, the key and signature sizes of the product scheme* $\Sigma_0 \otimes \Sigma_1$ *are:*

$$\begin{aligned} \mathrm{PK} &= \mathrm{PK}_0 \\ \mathrm{SK} &= \mathrm{SK}_0 + \mathrm{SK}_1 + \mathrm{PK}_1 + \mathrm{SIG}_0 + l \\ \mathrm{SIG} &= \mathrm{SIG}_0 + \mathrm{SIG}_1 + \mathrm{PK}_1 \end{aligned}$$

Theorem 6. *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 the forward-secure scheme* Σ_i *for* $i = 0, 1$*. Then the running times of the product scheme* $\Sigma_0 \otimes \Sigma_1$ *are:* [6]

$$\begin{aligned} \mathrm{KG} &= \mathrm{KG}_0 + \mathrm{KG}_1 + \mathrm{SG}_0 + \mathrm{UP}_0 + 2l^2 \\ \mathrm{SG} &= \mathrm{SG}_1 \\ \mathrm{VF} &= \mathrm{VF}_0 + \mathrm{VF}_1 \\ \mathrm{UP} &= (T_0(T_1 - 1)\mathrm{UP}_1 + (T_0 - 1)(l^2 + \mathrm{KG}_1 + \mathrm{SG}_0 + \mathrm{UP}_0))/(T_0T_1 - 1) \\ &\leq \mathrm{UP}_1 + (l^2 + \mathrm{KG}_1 + \mathrm{SG}_0 + \mathrm{UP}_0 - \mathrm{UP}_1)/T_1 \end{aligned}$$

4 The MMM Scheme

Here we present a new scheme obtained by the iterated application of the sum composition operation together with an asymmetric variant of the product construction. This scheme makes use of Merkle tree-type certification chains [10], combined with ideas from the binary tree scheme of Bellare and Miner [3]. 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 signing, verifying and update operations in the initial time periods is the same as if we had chosen a small bound on T in one of the previously known schemes. Performance degrades only slightly if a large number of time periods is used, and still remains competitive with all previously known schemes.

The idea is the following. We start from a regular, non-forward-secure, digital signature scheme S and build a forward-secure signature scheme $L = S^{\oplus}_{\log l}$ with l time periods, iterating the sum composition $\log l$ times. We then take the product of L with another scheme of the form $S^{\oplus}_i$, 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 $S^{\oplus}_i$

[6] As explained in Section 1.2, we assume that a call to a hash function or PRG takes time $O(l^2)$. Also, the update time here is the result of amortized analysis; it represents the average case.

scheme. The twist here is that we use a different i for every leaf (see Figure 6), $S_0^{\oplus}$ 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 anyway to avoid exhaustive search attacks.

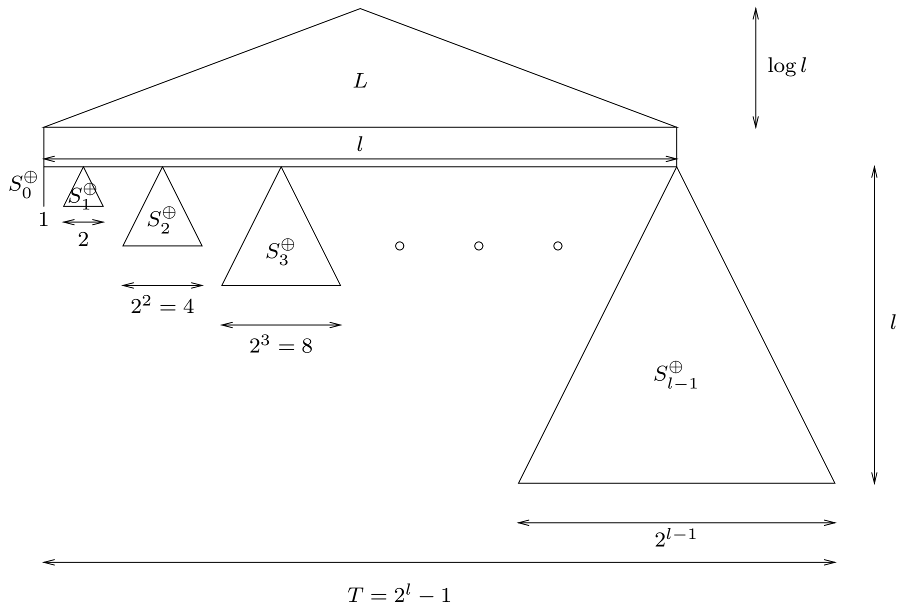

Fig. 6. The MMM construction, which uses an asymmetric product composition.

Security Analysis. Because of space limitations, a formal analysis of security is omitted in this extended abstract. The security of the MMM scheme follows from the security of the underlying composition operations, which we have shown to be security-preserving. However, because in the MMM scheme, 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, this number is always bounded by a polynomial, and the new scheme is guaranteed to satisfy asymptotic security.)

Performance Analysis. We now analyze the performance of the MMM scheme.[7]

[7] A summary of its performance when instantiated with the GQ scheme was given in Section 1.2.

Key sizes. The public key is just a hash value (l bits), while the secret key has roughly the same size as a digital signature: $O(k + (\log l + \log t)l)$ bits.

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.

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

Verification time. Verification consists of 2 regular signature verifications, and $\log l + \log t$ hash function evaluations. If the Guillou-Quisquater [7] signature scheme is used, this is $4k^2 l + (\log l + \log t)l^2 = O(k^2 l)$, i.e., just twice as much as the regular signature scheme.

Update time. Update consists of updating within the subtree S_i, or, between subtrees, it consists of generating the initial keys for the next subtree S_{i+1}, and updating the key of L. If we proceed in a straight forward manner, amortized analysis of the update time after t periods yields $O(\log t(k+l^2)+k^2 l)$. The worst case, however, is proportional to tlk^2, since in the beginning of a new epoch we need to generate keys for a new subtree S_{i+1} where $i = \log t$. As was discussed earlier, this update can be amortized across many time periods to achieve uniformly fast update, at the cost of larger secret keys. A similar amortization is described in detail by Merkle [10], so we give only a brief description here. The idea is to 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. The price paid for uniformly fast update is adding the space required to generate the initial keys for the next subtree to the size of the secret key. This results in an addition of $O(l \log^2 t + k \log l)$ to the secret key size.

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 recent scheme of [8], 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 [8] is implemented using their "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 [9], but not for the number theoretic constructions of [3, 2, 8].) For example, if the Rabin signature scheme is used, the verification time drops to $O(k^2)$, outperforming all non-generic schemes. The price in this example is making signature generation slightly slower, going from $O(k^2 l)$ 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.

5 Alternate Constructions

We used the sum and product composition operations discussed in Section 3 to generate the MMM construction above. Here, we briefly mention several other constructions possible using these operations, in order to highlight that different performance tradeoffs can be achieved by selecting the proper base schemes and the proper sequence of composition operations. (In fact, we can even achieve previously known schemes using our composition operations, as mentioned below.) In the full version of this paper [12], we analyze the following constructions in more detail:

- BM $\otimes$ BM (where BM denotes the main scheme of Bellare and Miner)
- the iterated product $\mathbf{S}^{\otimes}_{\log\log \mathbf{T}}$, exactly the binary tree scheme of Bellare and Miner
- the iterated sum $\mathbf{S}^{\oplus}_{\log \mathbf{T}}$
- $\mathbf{IR}^{\oplus}_{\mathbf{10}}$ (where IR denotes the standard construction of Itkis and Reyzin)

Acknowledgments

We thank Michel Abdalla and Leo Reyzin for helpful discussions regarding previous works, and comments on earlier drafts. We are also grateful to the anonymous reviewers for their detailed comments.

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. 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.

6. 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.
7. 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.
8. G. Itkis and L. Reyzin. *Forward-secure signatures with optimal signing and verifying.* In Advances in Cryptology - CRYPTO '01, LNCS **2139** (Aug. 2001), pp. 332–354.
9. H. Krawczyk. *Simple forward-secure signatures from any signature scheme.* In Seventh ACM Conference on Computer and Communications Security (Nov. 2000), pp. 108–115.
10. R. C. Merkle. *A certified digital signature.* In Advances in Cryptology - CRYPTO '89, (Aug. 1989), pp. 218–238.
11. S. Micali. *A secure and efficient digital signature algorithm.* Technical Report MIT/LCS/TM-501, Massachusetts Institute of Technology, March 1994.
12. T. Malkin, D. Micciancio and S. Miner. *Efficient generic forward-secure signatures with and unbounded number of time periods.* Full version of this paper, available at http://eprint.iacr.org/2001/034/.
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.