Efficient Generic Forward-Secure Signatures and Proxy Signatures

Basel Alomair, Krishna Sampigethaya, and Radha Poovendran

Network Security Lab. (NSL)
Electrical Engineering Department
University of Washington
{alomair,rkrishna,rp3}@u.washington.edu

Abstract. We propose a generic method to construct forward-secure signature schemes from standard signature schemes. The proposed construction is more computationally efficient than previously proposed schemes. In particular, the key updating operation in the proposed scheme is orders of magnitude more computationally efficient than previous schemes, making it attractive for a variety of applications, such as electronic checkbooks. Another advantage of our proposed scheme is the ability to be easily extended to proxy signature schemes. We define two notions of forward-security in the proxy signature setup, namely, strong forward-secure proxy signatures and weak forward-secure proxy signatures. We then describe a construction of a scheme that satisfies the strong forward-secure proxy signature property.

Keywords: Digital signatures, forward-security, proxy, efficiency.

1 Introduction

1.1 Background

One of the biggest threats to the security of standard digital signatures is from the exposure of the secret (signing) keys. If the secret key of an authorized user is exposed, an adversary with access to the exposed key can forge signatures that are indistinguishable from the signatures of the authorized user. Furthermore, all the signatures of the authorized user become repudiable, even if they have been generated before the key exposure. In order to minimize the damage caused by key exposure, the notion of forward-security in digital signatures was put forth by Anderson in [2] and formalized by Bellare and Miner in [3]. In forward-secure signature schemes (FSS), although an adversary with access to exposed keys can generate valid signatures, the validity of signatures generated prior to the key exposure will remain intact. Consequently, forged signatures with past dates are distinguishable from valid signatures.

A forward-secure signature scheme is a key evolving scheme [3]. In previously proposed schemes [3,1,13,11,5,14,18], time is divided into disjoint intervals, say T periods $t_1, t_2, ..., t_T$; each period t_i has a secret key, while the public key remains the same. At the end of each interval, a new secret key is generated and the secret key corresponding to the previous interval is deleted [3]. Hence, FSS are time dependent; that is, FSS must be correlated in some way to the time when the signature is generated. On the other hand, a verifier of an FSS must also have a mechanism to verify that the signature generated during interval t_i is uniquely related to the secret key that is valid at t_i. To ensure forward-secrecy, however, it is required that old secret keys cannot be computed by unauthorized users based on the knowledge of present or future keys.

As pointed out in [3], it is trivial to design FSS if we allow the size of the registered[1] secret and/or public keys, or the size of the signatures to grow proportionally with the number of intervals T. It is impractical, however, to assume users' ability to register an arbitrary number of keys from the Certificate Authority (CA) [3]. Except for [2], all proposed schemes avoid such an impractical assumption [3,1,13,11,5,14,18].

In the existing literature, there are two major classes of approaches to design FSS. The first approach is to design schemes based on specific number theoretic assumptions [3,1,13,11,5]. The second approach is to apply a generic construction to standard signature schemes [2,3,14,18]. Generic schemes have the advantage of provable-security assuming secure standard signature schemes exist [7]. On the other hand, specific schemes can only be proven secure in the random-oracle model[7]. Another advantage of generic schemes is that they exhibit the added flexibility to be instantiated from different standard signature schemes. This gives generic schemes the room for trading off computational and storage efficiencies by using base schemes with different performance characteristics, rather than being bound to the properties of a specific base scheme [7].

An unsolved problem in previously proposed schemes [3,1,13,11,5,14,18] is the validity of signatures generated within the key exposure interval. Obviously, all signatures generated before the key exposure but within the same interval will be repudiable, since the same key is used throughout the entire interval. Therefore, the design of interval lengths can be a nontrivial task. The longer the interval, the more the signatures generated with the same key, hence, violating the whole idea behind forward-security in digital signatures. On the other hand, shorter intervals will result in a more frequent key updates, even if no signature has been generated during the intervals.

In this paper, we propose a generic construction, that can be applied to any standard signature scheme based on the discrete logarithm problem (DLP)[2], to compose an FSS scheme. Our treatment of forward-security is different; instead of corresponding keys with time intervals, our keys are tied with signatures. That is, each key is used for one and only one signature; after every signature generation, the key is updated independent of time. In our scheme, the key update algorithm is orders of magnitude more computationally efficient than previously proposed schemes. Furthermore, computational and storage efficiencies are very competitive in all parts of the scheme, compared to other generic schemes in [14,18] and achieves better performance overall (Tables 2 and 3).

Another advantage of our scheme is that its extension to proxy signatures is straightforward and intuitive. We define two levels of forward-security in proxy signatures, and propose a generic construction of *forward-secure proxy signature schemes* that achieves the stronger level of security. To the best of our knowledge, the construction of forward-secure proxy signature schemes has not been addressed in the literature.

[1] Registered keys are the ones authenticated by the Certificate Authority.

[2] In fact, the construction can be extended to non-DLP based signature by slight modifications.

1.2 Related Work

In [2], Anderson extended the idea of forward-security in session key exchange protocols introduced in [10,8] to the context of digital signatures. Since then, many schemes have been proposed to design FSS where the parameters sizes (registered private/public keys and signatures) do not depend on the total number of intervals T. These proposals can be divided into two categories: schemes based on specific number theoretic assumptions, and generic schemes constructed from a standard signature scheme as a building block.

Number theoretic schemes. Bellare and Miner [3] formalized the problem introduced by Anderson [2] and suggested the first solution. Their proposed scheme was based on the hardness of factoring the Blum-Williams integers. Although their scheme does not require the number of secret or public keys, nor the length of the signature to grow with the number of intervals T, it does, however, have rather long keys. Other number theoretic schemes appeared later including [1,13,11,5]. Abdalla and Reyzin [1] improved on the scheme of [3] by shortening the length of the keys. Kozlov and Reyzin [13] proposed an FSS with fast key update. Their proposed scheme requires only one modular exponentiation to update the keys; hence, allowing for shorter intervals. Itkis and Reyzin [11] proposed an FSS that requires only two modular exponentiations with short exponents for signing and verifying. Boyen *et al.* [5] proposed a scheme where the update can be performed on encrypted keys as a second layer of security.

Generic approach schemes. A different approach to designing FSS is to use existing standard digital signature schemes as building blocks. Anderson [2] proposed the first scheme, where the signer is required to have a registered secret key for each interval, thus having a secret key size linear in T. Bellare and Miner [3] introduced the binary tree scheme to reduce the required number of registered secret keys to $O(\log T)$. Their improvement came with the cost of longer signatures and longer verification time (both are increased by a factor $O(\log T)$). As argued by Bellare and Miner [3], however, signature schemes with one of the parameters (registered secret/public keys, or signature sizes) growing proportionally with T are considered impractical.

The first practical generic construction was proposed by Krawczyk [14]. In the initialization phase of [14], the signer, using a single pair of registered secret/public keys, generates a total of T certificates, one for each period. The *certificate chain* of length T need not be secret. However, it must be available for the signer to generate valid signatures. If the certificate corresponding to time interval t_i is lost, no signature can be generated during that interval. The other generic scheme was the one proposed by Malkin *et al.*, usually referred to as the MMM scheme [18]. The MMM scheme makes use of Merkle tree-type certification chains combined with ideas from the binary tree scheme of [3]. In the MMM scheme, instead of generating T certificates in the initialization phase, the scheme is divided into subtrees. At the end of each subtree, $\log t_i$ secrets are generated, where t_i denotes the time period. Hence, reducing the secret key storage from $O(T)$ to $O(\log T)$.[3] This reduction, however, is paid for in the key

[3] In the worst case scenario, $i = T$.

update algorithm as their update time is increased by a factor of log t_i compared to [14]. Moreover, unlike the certificate chain [14], their keys must remain secret.

In a different but related work, Ma and Tsudik [17] proposed the forward secure sequential aggregate (fssagg) authentication schemes.

1.3 Our Results

Compared to previously proposed generic FSS [2,14,18] and the tree based scheme of Bellare and Miner [3], our generic construction method exhibits desirable properties, such as:

Time independence. All previously proposed schemes, including number theoretic based schemes, assign keys to time intervals. Except for the MMM scheme [18], all other schemes require the total number of intervals T to be predefined. Thus, the design of short intervals will exhaust the set of private keys rather quickly. On the other hand, the design of longer intervals leads to signing more messages with the same key, violating the principle of FSS.

Although, the MMM scheme [18] does not require the total number of time intervals to be predefined, the efficiency of some of its operations[4] is dependent upon the interval when the signature is generated. That is, the scheme will become less efficient with time. Therefore, the choice of shorter intervals will drive the scheme to be less efficient relatively rapidly.

This problem is overcome in our scheme by demanding that key update is performed immediately after every signature generation, rather than the end of the time interval.

Flexibility. In addition to the flexibility granted by the ability to use different standard schemes as building blocks, our scheme provides a flexibility of its own. It provides options to trade-off between space and computational efficiencies. For example, some parameters evaluated in the key generation stage can be stored in the system to make key updating and signing more computationally efficient. Moreover, some parameters are unrestricted and can be generated in either one of two operations. For example, a modular exponentiation operation can be performed either in the key update operation or the signature generation operation, trading off computational efficiencies in the two operations, depending on application.

Computational efficiency. As a direct consequence of the requirement that key update is performed immediately after signature generation, the key update operation must be performed efficiently. For that, we propose a key update algorithm that is orders of magnitude more efficient than the previous generic schemes [14,18], as can be found in Tables 2 and 3. Furthermore, the proposed scheme achieves excellent overall performance. In [18], Malkin *et al.* provide a comparison of number theoretic approaches [3,1,13,11] with their MMM scheme, showing how the MMM scheme outperforms these number theoretic approach schemes. Since the MMM scheme is arguably the most efficient generic scheme [7,16], we quantify, in Tables 2 and 3, the performances

[4] See Table 2.

of our proposed scheme against other generic construction based schemes, including the MMM scheme, showing how our proposed scheme outperforms those earlier schemes.

Applicability to proxy signature schemes. Our proposed scheme can be easily applied to proxy signature schemes. In fact, as discussed in Section 4, the implementation of forward-secure proxy signature schemes based on our proposed scheme is straightforward. The extension of generic construction to proxy signature schemes has not been addressed in any of the previous proposals for FSS.

The rest of the paper is organized as follows. In Section 2 we review some preliminaries and basic concepts that will be used in the subsequent sections. We present our generic construction in Section 3. In Section 4 we describe the construction of forward-secure proxy signature scheme. We conclude our paper in Section 5.

2 Preliminaries

Throughout the rest of the paper we assume the existence of public key infrastructure, where users have registered private and public key pair (x, y), where x represents the private key and y represents the public key. Depending on context, the term signing key is used interchangeably with the terms private or secret key. Similarly, the terms public and verifying key are used synonymously.

2.1 Definitions

A DLP-based standard signature scheme is defined as follows:

Definition 1 (Standard Digital Signature Scheme). *A standard digital signature scheme* SS= $(\mathcal{P}, \mathcal{K}, \mathcal{S}, \mathcal{V})$*, with* $\mathcal{P}, \mathcal{K}, \mathcal{S}$*, and* $\mathcal{V}$ *being polynomial-time algorithms with the following functionalities.*

1. $\mathcal{P}$ *is a randomized parameter-generating algorithm that, on input* 1^k*, where* k *is a security parameter, outputs a description of a multiplicative group* $\mathcal{G}$*, a generator* g*, and a description of a one-way hash function. These parameters are assumed to be publicly known.*
2. $\mathcal{K}$ *is a randomized key-generating algorithm that takes the output of* $\mathcal{P}$ *as input and outputs a pair of keys* (x, y)*, where* x *is a secret key and* y *is the corresponding public key.*
3. $\mathcal{S}$ *is a possibly randomized signing algorithm that takes as input a message* $M \in \{0, 1\}^*$*, a secret key* x*, and possibly a set of parameters* λ*. Depending on application, no set of parameters might be needed, in which case* λ *is taken to be the empty set. The algorithm outputs a signature* σ *on the message* M*.*
4. $\mathcal{V}$ *is a deterministic verification algorithm that takes as input* (M, y, σ)*, such that:*

$$\mathcal{V}(M, y, \sigma) = \begin{cases} 1, & if \;\; \sigma = \mathcal{S}(M, x, \lambda) \\ 0, & otherwise \end{cases} . \tag{1}$$

Equation (1) demands that the verification algorithm $\mathcal{V}$ *outputs* 1 *only if the signature* σ *on message* M *is generated using the secret key* x *corresponding to the public key* y*. Otherwise put, the verification algorithm* $\mathcal{V}$ *outputs* 1 *only if the signature is valid.*

2.2 Security Parameters

To be able to provide fair comparison for different schemes we need to quantify the cost of performing different operations in each scheme. Time and space complexity is used here to provide such a basis for comparison. For that, as in [18], we define two different security parameters.

l : a security parameter such that performing an exhaustive search over l-bit sequences is infeasible. This is the security parameter of conventional symmetric cryptography. We assume the output of cryptographic hash function and the size of secret keys in signature schemes are of size l.

k : a security parameter such that the k-bit composite integers are hard to factor and such that the discrete logarithm problem modulo a k-bit prime is believed to be hard. We assume that the size of public keys in signature schemes are of size k.

Distinguishing between the two parameters is important, because symmetric key cryptography is much more efficient than asymmetric. The factoring problem can be solved in sub-exponential time $\exp(k^{\frac{1}{3}} \log^{\frac{2}{3}} k)$, thus, asymptotically one might need $k \approx O(l^3)$ [18].

Although more efficient multiplication algorithms exist, for simplicity we will assume that multiplying two integers a and b is performed in $O(\text{size}(a)\ \text{size}(b))$ time and modular exponentiation is performed in $O(\text{size}(\textit{exponent})\ \text{size}(\textit{modulos})^2)$ time [6]. Throughout the rest of the paper, we will ignore the cost of integer addition and integer comparison.

3 Proposed Generic Construction of Forward Secure Signature Schemes

3.1 The Proposed Scheme

In this section we describe our generic construction method to design FSS. The key point for the forward-security of our scheme is allowing the signer to possess *two pairs of registered keys*. Allowing the signer to possess two pairs of registered keys results in a simple design of FSS that is computationally more efficient than previously proposed schemes. Instead of generating T certificates as in [14], or a secret for every tree leave as in [18], our scheme is based on the generation of a *forward-security chain* R. The forward-security chain is not required to generate signature, as opposed to [14], nor is it required to be kept secret, as opposed to [18]. In fact, it need not be stored in the system and signatures can still be generated and verified. Another advantage of our scheme and the scheme of [14], over the MMM scheme, is that the chain R and the T certificates are generated off-line in the key generation phase.

To describe our construction method, let SS=($\mathcal{P}, \mathcal{K}, \mathcal{S}, \mathcal{V}$) be a standard digital signature scheme as in Definition 1. Based on SS, the constructed forward-secure signature scheme is FSS=($\mathcal{P}, \mathcal{K}, \mathcal{FKG}, \mathcal{FS}, \mathcal{FV}, \mathcal{KU}$), where $\mathcal{P}, \mathcal{K}, \mathcal{FGK}, \mathcal{FS}, \mathcal{FV}$, and $\mathcal{KU}$ are polynomial-time algorithms. The algorithms $\mathcal{P}$ and $\mathcal{K}$ are exactly the same as in the base scheme. We further require that the algorithm $\mathcal{K}$ is run twice so the signer will possess two pairs of registered private and public keys (x_1, y_1) and (x_2, y_2). The forward-secure key generation algorithm $\mathcal{FGK}$, the forward-secure signing algorithm $\mathcal{FS}$, the forward-secure verifying algorithm $\mathcal{FV}$, and the forward-secure key update algorithm $\mathcal{KU}$ are described in detail below.

Key Generation. On input of a security parameter l, the user generates a prime p and a prime q that divides $p-1$, such that $q \geq 2^l$. The user picks an element $g \in \mathbb{Z}_p^*$ of order q, and selects a hash function $h : \{0,1\}^* \rightarrow \mathbb{Z}_q^*$. The parameters p, q, g, and h are assumed to be publicly known.[5] With the above public parameters and the total number of signatures for the forward-secure scheme T in hand, the signer generates the forward-security chain $R = (r^{(1)}, r^{(2)}, ..., r^{(T)})$, where each $r^{(i)}$ corresponds to i^{th} signature. To start, the signer generates a integer $k^{(1)}$ picked randomly from the multiplicative group $\mathbb{Z}_q^*$. The value of $r^{(1)}$ is then computed from $k^{(1)}$ as:

$$r^{(1)} = g^{k^{(1)}} \ mod \ p. \tag{2}$$

Using the one-way hash function h, the signer continues to construct a chain of $k^{(i)}$'s:

$$k^{(i)} = h(k^{(i-1)}), \tag{3}$$

of length T. For each $k^{(i)}$ the corresponding $r^{(i)}$ is computed as in equation (2) and the forward-security chain R is constructed as an ordered tuple of the $r^{(i)}$'s. Figure 1 illustrates an implementation of the key generation phase.

The function h in equation (3) must be a one-way function so that evaluating $k^{(i-1)}$ from $k^{(i)}$ can be assumed infeasible. Moreover, by the discrete logarithm assumption, computing $k^{(i)}$ using the knowledge of $r^{(i)}$ is infeasible.

```
Algorithm $\mathcal{FKG}(T)$
  $k^{(1)} \xleftarrow{R} \mathbb{Z}_q^*$;
  $r^{(1)} \leftarrow g^{k^{(1)}} \ mod \ p$;
  For $i = 2, ..., T$ do
    $k^{(i)} \leftarrow h(k^{(i-1)})$;
    $r^{(i)} \leftarrow g^{k^{(i)}} \ (mod \ p)$;
    Delete $k^{(i-1)}$;
  EndFor
  $R \leftarrow (r^{(1)}, r^{(2)}, r^{(3)}, ..., r^{(T)})$;
  Return $R$
```

After the forward-security chain R has been generated, the signer uses its first secret key x_1 to sign the chain (using any secure standard signature scheme.) The secret key x_1 is only needed to sign the chain R in the key generation phase and should not be stored in the system to ensure forward-secrecy. Otherwise, an adversary breaking into the system can forge a signature for any R. Note that the only parameter that the signer

[5] This setup is the same as in the Schnorr signature scheme [23]. For different standard signature schemes, setup varies according to the used standard scheme.

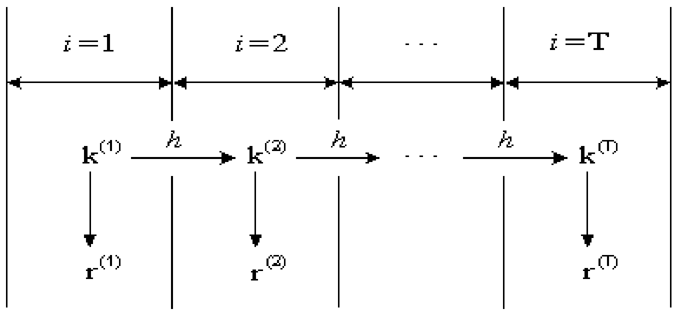

Fig. 1. The forward-security chain generation: Secret key for a given signature is a hash of the secret key for the previous signature

is required to store after the completion of the key generating phase is the value of the random number $k^{(1)}$ and the private key x_2. The chain R is used to provide forward-security and it is not required for signatures generation. Observe that the key generation is performed only once during the lifetime of the FSS, and it is performed off-line.

Signature Generation. To sign the i^{th} message, the signer uses its second secret key x_2 and the value of $k^{(i)}$ to run the signing algorithm $\mathcal{FS}$. The signer calls the base signing algorithm with $k^{(i)}$ passed as a parameter.

$$s = \mathcal{S}(M, x_2, \lambda = k^{(i)}), \tag{4}$$

where $\mathcal{S}$ represents the signing algorithm corresponding to the standard signature scheme used as a building block. The tuple $\sigma = (i, s, r^{(i)})$ comprises the signature on message M.

```
Algorithm $\mathcal{FS}(M, x_2, i, k^{(i)})$
    $s \leftarrow \mathcal{S}(M, x_2, \lambda = k^{(i)})$;
    Return $\sigma = (i, s, r^{(i)})$
```

Note that, instead of generating a random k for every signature as in DLP-based signatures, such as El-Gamal [9] or Schnorr [23], the value $k^{(i)}$ is passed as a parameter in our scheme. Thus, our forward-secure signature generation is more computationally efficient than its base scheme. No previous generic FSS can be more efficient then its base scheme. Further improvement can be made, depending on the resources available for the signer. If computational efficiency is more important than storage, the signer can store R in the system. Storing R in the system will save the signer one modular exponentiation by passing $r^{(i)}$ as a parameter; an operation that requires $O(lk^2)$ time to be performed. This efficiency improvement, however, comes with a cost of $O(Tk)$ bits in storage.

Signature Verification. The verification algorithm $\mathcal{FV}$ is shown below. The verifier uses the signer's public key y_2 to verify the validity of the signature, using the standard verification algorithm $\mathcal{V}$. Then, the verifier runs the standard signature verification to verify the validity of the forward-security chain R using the signer's public key y_1.

```
Algorithm $\mathcal{FV}(M, y_2, \sigma, R, y_1, \sigma_R)$
  If $\mathcal{V}(M, y_2, \sigma) = 1$
    If $\mathcal{V}(R, y_1, \sigma_R) = 1$   and   $r_\sigma^{(i)} = r_R^{(i)}$
      Return 1
    Else
      Return 0
    EndIf
  Else
    Return 0
  EndIf
```

Observe that, without the second check, which is to check the validity of the chain R and if the value $r^{(i)}$ in the signature σ is equal to the authenticated $r^{(i)}$ in R, the verification algorithm is just the verification algorithm of the base scheme. That is, the scheme can be used as a standard scheme and, if needed (e.g., in case of a dispute), R can be used to ensure forward-security.

Key Update. After the (i-1)th signature has been generated, the signer updates the secret key $k^{(i-1)}$ by applying the one-way hash function, to get $k^{(i)}$. As soon as the value of $k^{(i)}$ has been computed, the value of $k^{(i-1)}$ must be deleted to ensure forward-security, as can be seen in algorithm $\mathcal{KU}$ below.

```
Algorithm $\mathcal{KU}(k^{(i-1)})$
  $k^{(i)} \leftarrow h(k^{(i-1)})$;
  Delete $k^{(i-1)}$
  Return $k^{(i)}$
```

The key update operation uses one application of the hash function, an operation that requires $O(l^2)$ in time. No previous FSS [3,1,13,11,5,14,18] can achieve the same performance, except for Anderson's proposal [2] which has been argued to be impractical. Even better, by storing the chain of $k^{(i)}$'s in the system, the key update can be performed in $O(1)$ time with the expense of $O(Tl)$ bits in storage, giving the system designer a choice to trade-off storage and computational efficiencies. Note that if the chain of $k^{(i)}$'s is stored, $k^{(j)}$ must be deleted immediately after the generation of the j^{th} signature to ensure forward security.

Table 1 illustrates our construction of an FSS based on the Schnorr signature [23].

Theorem 1. *If an adversary can break the forward-security of our scheme, she can solve the discrete logarithm problem or invert a one-way function.*

Proof. Compared to the underlying signature scheme, our scheme introduces the precomputation of the $k^{(i)}$'s and the forward-security chain R. Given the security of the underlying scheme, the adversary needs to know the $k^{(i)}$ corresponding to the i^{th} signature to forge a valid signature. If $k^{(i)}$ has already been deleted from the system, the

Table 1. Applying the proposed construction method to design an FSS based on the Schnorr signature scheme [23]. The key generation $\mathcal{FKG}$, signature generation $\mathcal{FS}$, signature verification $\mathcal{FV}$, and the key updating $\mathcal{KU}$ algorithms are summarized.

```
Algorithm $\mathcal{FKG}(T)$
  $k^{(1)} \xleftarrow{R} \mathbb{Z}_q^*$
  $r^{(1)} \leftarrow g^{k^{(1)}}$ $mod\ p$;
  For $i = 2, ..., T$ do
    $k^{(i)} \leftarrow h(k^{(i-1)})$;
    $r^{(i)} \leftarrow g^{k^{(i)}}$ $(mod\ p)$;
    Delete $k^{(i-1)}$
  EndFor
  $R \leftarrow (r^{(1)}, r^{(2)}, r^{(3)}, ..., r^{(T)})$;
  Return $R$
```

```
Algorithm $\mathcal{FS}(M, x_2, i, k^{(i)})$
  $r^{(i)} \leftarrow g^{k^{(i)}}$ $mod\ p$;
  $s \leftarrow h(M, i, r^{(i)})x_2 + k^{(i)}$ $mod\ q$;
  Return $\sigma = (i, s, r^{(i)})$
```

```
Algorithm $\mathcal{FV}(M, y_2, \sigma, R, y_1, \sigma_R)$
  If $r^{(i)} \equiv y_2^{h(M,i,r^{(i)})} g^s$ $mod\ p$
    If $\mathcal{V}(R, y_1, \sigma_R) = 1$ and $r_\sigma^{(i)} = r_R^{(i)}$
      Return 1
    Else
      Return 0
    EndIf
  Else
    Return 0
  EndIf
```

```
Algorithm $\mathcal{KU}(k^{(i-1)})$
  $k^{(i)} \leftarrow h(k^{(i-1)})$;
  Delete $k^{(i-1)}$
  Return $k^{(i)}$
```

adversary can compute $k^{(i)}$ either from $r^{(i)}$ by solving the DLP or from present $k^{(j)}$, $j > i$, by inverting the one-way hash function. $\square$

3.2 Performance Analysis

In this section we analyze the performance of our scheme.

Key generation. The key generation requires the user to generate a total of T $k^{(i)}$'s, one for each signature. Each $k^{(i)}$ is obtained from $k^{(i-1)}$ by one application of the hash function. Each application of the hash function is performed in $O(l^2)$ time. For each $k^{(i)}$, the corresponding $r^{(i)}$ is computed using one modular exponentiation. Each computation of $r^{(i)}$ is performed in $O(lk^2)$ time. Therefore, the entire key generation operation is performed in $O(T(l^2 + lk^2))$ time.

Signature generation. Our forward-secure signature generation is the signature generation of the standard scheme with $k^{(i)}$ passed as a parameter. If the Schnorr signature scheme is used, the signature generation of our scheme is performed in $O(lk^2 + 2l^2)$ time.

Table 2. Comparing the asymptotic performances of the generic constructions of Krawczyk [14], MMM [18], and our scheme, when the Schnorr signature scheme [23] is used as a building block

	Krawczyk	MMM	**Our Scheme**
Key update time	$2l^2 + lk^2$	Tk^2l	l^2
Signature time	$3l^2 + lk^2$	$3l^2 + lk^2$	$2l^2 + lk^2$
Verifying time	$2(2lk^2 + l^2)$	$2(2lk^2 + l^2) + (\log l + \log T)l^2$	$2(2lk^2 + l^2)$
Signature size	$\log T + 2l + 4k$	$(\log l + \log T + 2)l + 4k$	$\log T + l + k$
Key gen time	$(5l^2 + 2lk^2)T$	k^2l^2	$(l^2 + lk^2)T + lk^2 + 3l^2$
Secret key size	l	$k + l(\log l + \log T)$	$2l$
Public key size	k	l	$2k$

Signature verification. The verification of our scheme is the same as in the standard scheme with the added verification of the forward-security chain R and the check that $r^{(i)}$ in the signature is equal to $r^{(i)}$ in R. If Schnorr signature scheme is used, the verification operation can be performed in $O(4lk^2 + 2l^2)$. Observe, however, the verification of more than one signature requires only a single verification of the forward-security chain R.

Key update. After the generation of every signature, the key $k^{(i-1)}$ is updated to get $k^{(i)}$ and the old key is deleted. The hashing operation can be performed in $O(l^2)$ time. Although this key update operation outperforms all previous schemes, it can be made even more efficient. If the user can store the chain of $k^{(i)}$'s generated during the key generation phase, the key update can be performed in $O(1)$ time.

Key sizes. Since the signer possesses two pairs of registered keys, the size of the secret key is $2l$ bits, while the size of the public key is $2k$ bits.

Signature size. The only extra parameter in the signature of our scheme, compared to the standard scheme, is the inclusion of the signature's number i. Resulting in an extra $\log i$ bits in the size of the signature. If Schnorr signature is used, the resulting signature size will be $O(l + k + \log i)$ bits.

3.3 Applicability

The properties and performances of all parts of our scheme make it attractive for a variety of applications. Our focus will be electronic checkbooks (e-checkbooks) which was our primary motivation. Just like a regular checkbook, an e-checkbook usually contains multiple checks. Each e-check is meant to be signed by the checkbook holder and verified by the bank issuing the e-checkbook.

The signature number in our scheme is mapped to the check serial number in the e-checkbook application. When a scheme with inefficient key update algorithm is used and the user's e-checkbook is stolen, an unauthorized user with access to the e-checkbook can forge a signature on the last signed e-check that is indistinguishable from the authorized signature, making the last check signed by the authorized user repudiable. Therefore, a scheme with very efficient key update algorithm, like our scheme,

Table 3. Comparing the performances of the generic constructions of Krawczyk [14], MMM [18], and our scheme, when the Schnorr signature scheme [23] is used as a building block and for $k = 1024$, $l = 160$, and $T = 1000$. $(*)$ this operation is done only once for the lifetime of the forward secure scheme and is performed offline.

	Krawczyk		MMM		**Our Scheme**
	Absolute	Relative	Absolute	Relative	Absolute
Key update time	1.7×10^8	6800	1.7×10^{11}	6800000	2.5×10^4
Signature time	1.7×10^8	1	1.7×10^8	1	1.7×10^8
Verifying time	6.7×10^8	1	6.7×10^8	1	6.7×10^8
Signature size	4.4×10^3	3.67	7.2×10^3	6	1.2×10^3
Key gen time	3.4×10^{11}	2	2.7×10^{10}	$-6.3(*)$	1.7×10^{11}

is of most importance to the application of e-checkbook. Furthermore, since signatures are meant to be verified by the bank, the forward-security chain R can be stored in the bank's database; thus, no need to worry about the exchange of the chain R. Moreover, the verification of the chain R needs to be performed only once.

3.4 Comparison to Other Schemes

In this section, we compare the performance of our scheme with the Krawczyk [14] and Malkin *et al.* [18] when the Schnorr signature scheme is used for construction. Table 2 shows the asymptotic performances of the three schemes in the worst case scenario (when $t_i = T$ for the schemes in [14,18], and $i = T$ in our scheme). If $k \approx O(l^3)$, as suggested in [18], the three schemes are asymptotically comparable, except for the key update operation where the performances are $O(l^7)$, $O(Tl^7)$, and $O(l^2)$ for the Krawczyk, MMM, and our scheme respectively. Thus, supporting our claim that our key update algorithm is orders of magnitude more efficient.

Table 3 compares the performances when $k = 1024$, $l = 160$, and $T = 1000$. The values of k and l are the typical values in Schnorr signature scheme. The columns corresponding to the Krawczyk and the MMM schemes are split into two halves. The first half shows the performance of the corresponding scheme where the second half shows the normalized performances when our scheme is used as a reference point. Positive values represent the relative advantage of our scheme, while negative values represent the advantage of their schemes over ours. Since storage is cheap in today's technology, the non-secret storage of the T certificates in the Krawczyk scheme and the chain R in our scheme are ignored.

In the next section, we show how to construct forwrad-secure proxy signature schemes.

4 Forward-Secure Proxy Signature Scheme

4.1 Background

In the previous section, we showed how to construct an FSS with the requirement that the signer possesses two pairs of registered keys. In this section, we show how the same idea can be extended to construct a forward-secure proxy signature scheme (FSPS) from any secure proxy signature scheme, without the requirement that the signer possesses two pairs of registered keys. This requirement is avoided because, in the proxy signature setup, there are two legitimate users, namely, the proxy designator and the proxy signer, and each one of them is assumed to possess a pair of registered private and public keys, (x_a, y_a) and (x_b, y_b), respectively. For historical reasons the proxy designator and signer are called Alice and Bob respectively. In proxy signature schemes, Alice delegates her signing capability to Bob. The idea of digital proxy signatures was first introduced by Mambo *et.al.* [20]. Many of the proposed proxy signature schemes appeared in the literature are based on the following concept: Alice has a pair of keys (x_a, y_a), where x_a and y_a represent the secret and public keys, respectively. To delegate her signing power to the Bob, Alice generates a warrant describing Bob's authorities to sign messages on her behalf. The warrant is then signed by Alice (using a standard signature scheme) and sent to Bob. After checking the validity of the signature, Bob combines Alice's signature with his secret key x_b to generate a *proxy* signing key x_p. Bob uses the *proxy* signing key x_p to sign messages on behalf of Alice using a standard signature scheme. To validate a proxy signature, the verifier computes the public key y_p corresponding to the proxy secret key x_p (usually a function of Alice's and Bob's public keys; that is, $y_p = f(y_a, y_b)$) and use the corresponding standard signature verification algorithm to verify the signature.

Many proxy signature schemes have appeared in the literature [21,22,12,15,4]. As noted by Wang [24], little effort has been made towards the design of FSPS. In most of the existing proxy signature schemes, Bob generates a proxy key x_p, and the same key will be used for proxy signing throughout the signature delegation period, without any modifications. Hence, exposure of the proxy key x_p will enable a forger to generate proxy signatures that are indistinguishable from the legitimate signatures. We propose, to the best of our knowledge, the first design of an FSPS.

4.2 Definitions

It is important to distinguish between two keys, namely, Bob's secret key x_b and the key generated by Bob to generate signatures on Alice's behalf x_p. Bob's key x_b is the one distributed by the CA and does not change. The other key x_p is generated by Bob and can be updated whenever Bob wishes.

Before describing the construction, we define two different notions of forward-security for proxy signatures; weak forward-secure proxy signature (WFSPS) schemes and strong forward-secure proxy signature (SFSPS) schemes.

Definition 2 (Weak Forward Secure Proxy Signatures). *A proxy signature scheme is said to be weak forward-secure if it is resilient to the exposure of the proxy key x_p.*

Definition 3 (Strong Forward Secure Proxy Signatures). *A proxy signature scheme is said to be strong forward-secure if it is resilient to the exposure of the proxy key x_p and the private key of the proxy signer x_b.*

The lack of appropriate definitions for forward security in proxy signatures led Malkin *et al.* to conclude that proxy signatures are by design forward secure [19]. Given our

Table 4. A forward-secure proxy signature scheme constructed by applying the proposed construction method to the proxy signature scheme in [12]. Assuming the forward-security chain has been generated successfully, the proxy key generation $\mathcal{PKG}$, proxy signature generation $\mathcal{PS}$, proxy signature verification $\mathcal{PV}$, and the proxy key updating $\mathcal{PKU}$ algorithms are summarized. The subscript a indicates parameters generated by the proxy designator and w represents the warrant describing the authority given to the proxy, as in the original scheme in [12]. x_b represents the registered secret key of the proxy signer, while x_p represents the key generated to sign messages.

```
Algorithm $\mathcal{PKG}(w, \sigma_a, k^{(i)}, i, x_b)$
  $r^{(i)} \leftarrow g^{k^{(i)}} \ mod\ p$
  $x_p^{(i)} \leftarrow h(i, r^{(i)})x_b + h(w, r_a)x_a + k_a \ mod\ q$
  Return $x_p^{(i)}$
```

```
Algorithm $\mathcal{PS}(M, x_p, k^{(i)})$
  $s_p \leftarrow h(M)x_p^{(i)} + k^{(i)} \ mod\ q$;
  Return $\sigma_M = (i, s_p, r^{(i)}, w, r_a)$;
```

```
Algorithm $\mathcal{PV}(y_a, y_b, M, \sigma_M, R, \sigma_R)$
  $y_p^{(i)} \leftarrow y_b^{h(i,r^{(i)})} y_a^{h(w,r_a)} r_a^{-1} \ mod\ p$;
  If $r^{(i)} \equiv (y_p^{(i)})^{h(M)} g^{s_p} \ mod\ p$
    If $\mathcal{V}(R, y_a, \sigma_R) = 1$ and $r_\sigma^{(i)} = r_R^{(i)}$
      Return 1
    Else
      Return 0
    EndIf
  Else
    Return 0
  EndIf
```

```
Algorithm $\mathcal{PKU}(k^{(i-1)})$
  $k^{(i)} \leftarrow h(k^{(i-1)})$;
  Delete $k^{(i-1)}$
  Return $k^{(i)}$
```

definitions above, under the condition that Bob generates a new x_p for every signature, proxy signatures are only weak forward-secure.

4.3 Our Proposed Strong Forward-Secure Proxy Signature Scheme

The idea here is the same idea for constructing regular FSS. For lack of space, we omit describing the details of the construction and outline the basic concept. The major difference between the standard proxy signature scheme and the forward-secure version is in the proxy and key initialization stage, which we describe below.

Proxy and key Initialization. The proxy key generation is an interactive protocol. To start the protocol, Alice decides the number of signatures T for the forward-secure proxy scheme. Upon receiving T, Bob runs the same algorithm $\mathcal{FKG}$ used in the construction of non-proxy forward-secure signature schemes to generate the forward-security chain R of length T. The forward-security chain R is then sent to Alice who signs R with her private key x_a and publishes the signed R.

For the i^{th} message M to be signed, the pair $(k^{(i)}, r^{(i)})$ is used by Bob to generate forward-secure proxy signatures with the private key x_p. The fact that R is signed by Alice ensures forward-security, even if Bob's system has been broken into.

Table 4 illustrates our construction of a forward-secure proxy signature scheme based on the provable secure scheme of Kim *et al.* [12]. The constructed scheme in Table 4 assumes that the proxy and key initialization has been performed successfully and the forward-security chain R is available for verifiers.

The security analysis of the forward-secure proxy scheme is similar to the non-proxy one. Assuming the standard proxy scheme is secure[6], provided that the forward-security chain R has been generated successfully, the forward-security is granted by the hardness of the discrete logarithm problem and the existence of one-way functions.

5 Conclusion

In this paper, we studied the two major design approaches of forward secure signature schemes, i.e. number-theoretic and generic construction based approaches. The generic construction schemes can be based on any underlying signature scheme, hence, offering more flexibility in terms of meeting a wide spectrum application requirements. We proposed a generic construction method to obtain a forward-secure signature scheme that is very efficient in parameter size and computation times. A quantitative comparison with related schemes showed that the proposed scheme is more computationally efficient in terms of signature verification, signature size, and key updating time.

Further, we identified that in the literature there was no explicit design of forward secure proxy signatures. Therefore, we first defined two notions of forward security in the context of proxy signatures, i.e. weak forward-secure and strong forward-secure proxy signatures. We then showed how the proposed generic construction method can be easily extended to any proxy signature scheme to obtain strong forward secure proxy signatures.

Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable comments.

References

1. Abdalla, M., Reyzin, L.: A new forward-secure digital signature scheme. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 116–129. Springer, Heidelberg (2000)
2. Anderson, R.: Two remarks on public key cryptology. In: Invited Lecture, ACM-CCS 1997 (1997)
3. Bellare, M., Miner, S.: A forward-secure digital signature scheme. In: Wiener, M.J. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 431–448. Springer, Heidelberg (1999)
4. Boldyreva, A., Palacio, A., Warinschi, B.: Secure Proxy Signature Schemes for Delegation of Signing Rights (2003), http://eprint.iacr.org/2003/096
5. Boyen, X., Shacham, H., Shen, E., Waters, B.: Forward-secure signatures with untrusted update. In: Proceedings of the 13th ACM conference on Computer and communications security, pp. 191–200 (2006)

[6] Kim *et al.* [12] is an example of a provable secure proxy signature scheme.

6. Buchmann, J.: Introduction to Cryptography. Springer, Heidelberg (2004)
7. Cronin, E., Jamin, S., Malkin, T., McDaniel, P.: On the performance, feasibility, and use of forward-secure signatures. Proceedings of the 10th ACM conference on Computer and communications security, 131–144 (2003)
8. Diffie, W., Oorschot, P.C., Wiener, M.: Authentication and authenticated key exchanges. Designs, Codes and Cryptography 2(2), 107–125 (1992)
9. Elgamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory 31(4), 469–472 (1985)
10. Günther, C.: An identity-based key-exchange protocol. In: Quisquater, J.-J., Vandewalle, J. (eds.) EUROCRYPT 1989. LNCS, vol. 434, pp. 29–37. Springer, Heidelberg (1990)
11. Itkis, G., Reyzin, L.: Forward-secure signatures with optimal signing and verifying. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 332–354. Springer, Heidelberg (2001)
12. Kim, S., Park, S., Won, D.: Proxy signatures, Revisited. In: Proceedings of the First International Conference on Information and Communication Security, pp. 223–232 (1997)
13. Kozlov, A., Reyzin, L.: Forward-Secure Signatures with Fast Key Update. In: Cimato, S., Galdi, C., Persiano, G. (eds.) SCN 2002. LNCS, vol. 2576, pp. 241–256. Springer, Heidelberg (2003) Revised Papers
14. Krawczyk, H.: Simple forward-secure signatures from any signature scheme. In: Proceedings of the 7th ACM conference on Computer and communications security, pp. 108–115 (2000)
15. Lee, B., Kim, H., Kim, K.: Strong proxy signature and its applications. In: Proc. of SCIS - Symposium on Cryptology and Information Security, pp. 603–608 (2001)
16. Libert, B., Quisquater, J., Yung, M.: Forward-secure signatures in untrusted update environments: efficient and generic constructions. In: Proceedings of the 14th ACM conference on Computer and communications security, pp. 266–275 (2007)
17. Ma, D., Tsudik, G.: Extended Abstract: Forward-Secure Sequential Aggregate Authentication. In: IEEE Symposium on Security and Privacy, 2007. SP 2007, pp. 86–91 (2007)
18. Malkin, T., Micciancio, D., Miner, S.: Composition and Efficiency Tradeoffs for Forward-Secure Digital Signatures. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, Springer, Heidelberg (2002)
19. Malkin, T., Obana, S., Yung, M.: The Hierarchy of Key Evolving Signatures and a Characterization of Proxy Signatures. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 306–322. Springer, Heidelberg (2004)
20. Mambo, M., Usuda, K., Okamoto, E.: Proxy Signatures: Delegation of the Power to Sign Messages. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences 79(9), 1338–1354 (1996)
21. Mambo, M., Usuda, K., Okamoto, E.: Proxy signatures for delegating signing operation. In: Proceedings of the 3rd ACM conference on Computer and communications security, pp. 48–57 (1996)
22. Petersen, H., Horster, P.: Self-certified keys-concepts and applications. Proc. Communications and Multimedia Security 97, 102–116 (1997)
23. Schnorr, C.: Efficient signature generation by smart cards. Journal of Cryptology 4(3), 161–174 (1991)
24. Wang, G.: Designated-verifier proxy signature schemes. In: Security And Privacy in the Age of Ubiquitous Computing: IFIP TC11 20th International Information Security Conference, Chiba, Japan, May 30-June 1, 2005 (2005)