Forward-Secure Signatures with Fast Key Update

Anton Kozlov Leonid Reyzin

Boston University Computer Science
111 Cummington St
Boston MA 02215 USA
{akozlov,reyzin}@cs.bu.edu

August 1, 2002

Abstract

In regular digital signatures, once the secret key is compromised, all signatures, even those that were issued by the honest signer before the compromise, will not be trustworthy any more. Forward-secure signatures have been proposed to address this major shortcoming.

We present a new forward-secure signature scheme, called KREUS, with several advantages. It has the most efficient Key Update of all known schemes, requiring just a single modular squaring. Our scheme thus enables more frequent Key Update and hence allows shorter time periods, enhancing security: fewer signatures might become invalid as a result of key compromise. In addition, the on-line component of Signing is also very efficient, consisting of a single multiplication. We precisely analyze the total signer costs and show that they are lower when the number of signatures per time period is small; the advantage of our scheme increases considerably as the number of time periods grows.

Our scheme's security relies on the Strong-RSA assumption and the random-oracle-based Fiat-Shamir transform.

1 Introduction

1.1 Prior work

The notion of forward-secure signatures was introduced in 1997 by Anderson [And97] to remove a serious limitation of regular digital signatures: once the secret key is lost (or stolen), *all documents* previously signed with that key become invalid. To reduce the damage, Anderson proposed to divide the validity time of a key pair into time periods. At the end of each time period, the signer derives the new secret key from the current one in a one-way fashion, and then securely erases the no-longer-needed current secret key. The public key, in contrast, remains unchanged during the life span of the key pair. This general approach ensures validity of all documents signed prior to the time period of compromise.

In recent years several forward-secure schemes were developed. Bellare and Miner [BM99] gave the formal definition of forward-secure signatures by extending the security definition for ordinary signatures of Goldwasser, Micali, and Rivest [GMR88]. They also proposed two schemes: one based on a tree-like structure of certificates using any ordinary signature scheme, and the other based on modifying the Fiat-Shamir [FS86] ordinary signature scheme. Following this work, many other forward-secure signature schemes were developed. The development has followed two paths: one was based on modifying specific, Fiat-Shamir-like, signature schemes, while the other used any ordinary signature scheme as a black box. Our scheme belongs to the first category, in which two other schemes (in addition to the one of [BM99]) are known. Abdalla's and Reyzin's scheme [AR00] is based on modifying Ong-Schnorr signatures [OS90], and shortens the secret and public keys of the Fiat-Shamir-based scheme of [BM99] (at the expense of Signing and Verifying time). Itkis' and Reyzin's scheme [IR01] is based on modifying Guillou-Quisquater [GQ88] signatures, and results in both short keys and efficient Signing and Verifying. We summarize the relevant performance attributes of these schemes in Table 1.

We note that forward-secure signature schemes in the second category (the ones using any ordinary signature scheme as a black box) are competitive, as well. Krawczyk's scheme [Kra00] is based on generating all of the certificates in advance in a pseudorandom manner. The most recent work by Malkin, Micciancio and Miner [MMM02] modifies the tree-based scheme of Bellare and Miner to achieve greater efficiency and remove the requirement (present in all prior schemes) that the number of time periods be fixed in advance, prior to Key Generation. These schemes are difficult to compare directly with schemes in the first category, because they depend heavily on how the ordinary signature scheme is instantiated. They can outperform schemes in the first category in some (but not all) parameters.

1.2 Our contribution

Our forward-secure digital signature scheme significantly improves the speed of Key Update algorithm: it becomes just a single modular squaring of the secret key, which is faster than in all previously known schemes. We therefore call our scheme KREUS for "Kozlov-Reyzin Efficient Update Signatures." Our Signing and Verifying algorithms, although slower than those of some other previous schemes, are also reasonably efficient. In particular, the on-line component of Signing (i.e., the part of the signing algorithm that can be performed only once the message is known) is extremely efficient, requiring less than a single multiplication.

Our key and signature sizes are small, and no extra storage is required. Similarly in spirit (and in technique) to [IR01], at the cost of storing $\log T$ extra values and increasing the Update time to $\log T$ modular squarings (this still more efficient than all prior schemes), we can improve the speed of the off-line component of Signing, as further detailed in Section 4. This is a surprising and novel application of the techniques of Jakobsson [Jak02] and Coppersmith and Jakobsson [CJ02] for traversing hash sequences.

Our construction was inspired by the work of Song in the area of forward-secure Group Signature schemes [Son01], that in turn is based on the ordinary Group Signature scheme by Ateniese, Camenisch, Joye, and Tsudik [ACJT00]. Since we do not need many features pertinent to group signatures (such as coalition resistance, anonymity, backward unlinkability, traceability, etc.), our scheme is more concise and simpler to implement. All we rely on is the proof of knowledge of Discrete Logarithm in a group of unknown size initially proposed by Fujisaki and Okamoto in [FO97].

We note that both our scheme and the scheme of [AR00] rely on the signer knowing some iterated square root of the public key. The scheme of [AR00] is much less efficient, however, because the signer needs to know a root of a much higher degree. We can use lower-degree roots because we use a different technique for proving knowledge of a square root, as further detailed in Section 3.

2 Definitions

2.1 Forward-secure digital signature schemes

The definitions of forward security are taken almost verbatim from [AR00], which in turn follows very closely the definitions of [BM99].

A forward-secure digital signature scheme is, first of all, a *key-evolving* digital signature scheme. A key-evolving signature scheme is very similar to a standard one. Like a standard signature scheme, it has Key Generation, Signing and Verification algorithms. The public key is left unchanged throughout the lifetime of the scheme, making the Verification algorithm very similar to that of a standard signature scheme. Unlike a standard signature scheme, a key-evolving signature scheme has its operation divided into time periods, each of which uses a different (but related) secret key to sign messages. The way these keys are updated is given by a public Key Update algorithm, which computes the secret key for the new time period based on that for the previous time period. The forward security comes, in part, from the fact that this Key Update function is one-way and, given the secret key for the current period, it is hard to compute any of the previously used secret keys. Let us now define more formally what a key-evolving digital signature scheme is and then define what it means for it to be forward-secure.

Definition 1 (Key-evolving signature scheme). A key-evolving digital signature scheme is a quadruple of algorithms, $FSIG = (\mathsf{FSIG.key}, \mathsf{FSIG.update}, \mathsf{FSIG.sign}, \mathsf{FSIG.verify})$, where:

- $\mathsf{FSIG.key}$, the Key Generation algorithm, is a probabilistic algorithm which takes as input a security parameter $k \in N$ (given in unary as 1^k) and the total number of periods T and returns a pair (SK_1, PK), the initial secret key and the public key;

- $\mathsf{FSIG.sign}$, the (possibly probabilistic) Signing algorithm, takes as input the secret key SK_j for the current time period j and the message M to be signed and returns a pair $\langle j, sign\rangle$, the signature of M for time period j;

- $\mathsf{FSIG.update}$, the (possibly probabilistic) Secret Key Update algorithm, takes as input the secret key for the current period SK_j and returns the new secret key SK_{j+1} for the next period;

- $\mathsf{FSIG.verify}$, the (deterministic) Verification algorithm, takes as input the public key PK, a message M, and a candidate signature $\langle j, sign\rangle$, and returns 1 if $\langle j, sign\rangle$ is a *valid* signature of M or 0, otherwise. We require $\mathsf{FSIG.verify}_{PK}(M, \mathsf{FSIG.sign}_{SK_j}(M)) = 1$ for every message M and time period j.

We also assume that the secret key SK_j for time period $j \leq T$ always contains both the value j itself and the value T of the total number of periods. Finally, we adopt the convention that SK_{T+1} is the empty string and that $\mathsf{FSIG.update}_{SK_T}$ returns SK_{T+1}. Since we are going to work in the random oracle model [BR93], all the above-mentioned algorithms would additionally have oracle access to a public hash function $H : \{0,1\}^* \to \{0,1\}^l$ for some security parameter l that is known to all the algorithms. The function H is assumed to be random in the security analysis.

Having defined a key-evolving signature scheme, we now define what it means for such a scheme to be forward-secure. We concern ourselves here with security only in the random oracle model, because this model is required for our scheme.

To define security, we need to specify the adversary. Besides knowing the user's public key PK, the adversary also gets to know the total number of time periods T and the current time period j. The adversary runs in three phases (it is allowed to preserve state information between phases). In the first phase, the *chosen message attack phase* (cma), the adversary has access to a signing oracle, which it can query to obtain signatures of messages of its choice with respect to the current secret key; we designate by q_{sig} the total number of signature queries. The adversary can also make queries to the random oracle H at any time; we designate by q_{hash} the total number of H-queries. At the end of each time period, the adversary can choose whether to stay in the same phase or switch to the *break-in phase* (breakin). In the break-in phase, which models the possibility of a key exposure, we give the adversary the secret key SK_j for the specific time period j it decided to break in. In the last phase, the *forgery phase* (forge), the adversary outputs a signature-message pair, that is, a forgery. The adversary is considered to be successful if it forges a signature of some *new* message (that is, not previously queried to the signing oracle) for some time period prior to j. In order to capture the notion of forward security of a key-evolving signature scheme FSIG = (FSIG.key, FSIG.update, FSIG.sign, FSIG.verify) more formally, let F be an adversary for this scheme. To assess the success probability of F breaking the forward security of FSIG, consider the following experiment:

```
**Experiment** F-Forge-RO(FSIG, $F$)
    Select $H$: $\{0,1\}^* \rightarrow \{0,1\}^l$ at random
    $(PK, SK_1) \overset{R}{\leftarrow} \mathsf{FSIG.key}^H(k, l, T)$
    $j \leftarrow 1$
    Repeat
        $d \leftarrow F^{H, \mathsf{FSIG.sign}^H_{SK_j}(\cdot)}(\mathsf{cma}, PK)$
         if $(d \neq \mathsf{breakin})$
            $SK_j \leftarrow \mathsf{FSIG.update}^H(SK_j); j \leftarrow j + 1;$
    Until $(d = \mathsf{breakin})$ or $(j = T + 1)$
    $(M, \langle b, sign \rangle) \leftarrow F^H(\mathsf{forge}, \mathsf{SK}_j)$
    If $\mathsf{FSIG.verify}^H_{PK}(M, \langle b, sign \rangle) = 1$ and $1 \leq b < j$
        and $M$ was not queried by $\mathsf{FSIG.sign}^H_{SK_b}(\cdot)$ in period $b$
        then return 1 else return 0
```

Finally, to define security, let FSIG = (FSIG.key, FSIG.update, FSIG.sign, FSIG.verify) be a key-evolving signature scheme, H be a random oracle and F be an adversary as described above. We now define $\mathbf{Succ}^{\mathsf{fwsig}}(\mathsf{FSIG}[k, l, T], F)$ as the probability that the experiment F-Forge-RO($\mathsf{FSIG}[k, l, T], F$) returns 1. Then the insecurity of FSIG is the function

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

where the maximum is taken over all adversaries F making a total of at most q_{sig} queries to the signing oracle across all the stages and for which the running time of the above experiment is at most t and at most q_{hash} queries are made to the random oracle H. The smaller the insecurity function, the more forward-secure the scheme is. To obtain an asymptotic definition of security, one would polynomially relate k, l, and T to a single security parameter, and define the scheme to be forward-secure if the insecurity function is negligible in the security parameter.

2.2 Strong-RSA Assumption

We use essentially the same Strong-RSA assumption as [IR01], and hence quote it from there almost verbatim.

The assumption was first introduced independently in [BP97] and [FO97], and postulates that it is hard to compute *any* root of a fixed value modulo a composite integer. More precisely, the Strong-RSA assumption states that it is intractable, given n that is a product of two primes and a value α in Z_n^*, to find $\beta \in Z_n^*$ and $r > 1$ such that $\beta^r \equiv \alpha \pmod{n}$.

Just like [IR01], we modify the assumption in two ways. First, we restrict ourselves to the moduli that are products of so-called "safe" primes (a safe prime is one of the form $2q + 1$, where q itself is a prime). Note that, assuming safe primes are frequent, this restriction does not strengthen the assumption. Second, we upperbound the permissible values or r by 2^l, where l is a security parameter for our scheme (in an implementation, l will be significantly shorter than the length k of the modulus n; the sole difference between our assumption and the one in [IR01] is that [IR01] upperbound r by 2^{l+1} instead of 2^l).

More formally, let A be an algorithm. Consider the following experiment.

```
Experiment Break-Strong-RSA$(k, l, A)$
    Randomly choose two primes $q_1$ and $q_2$ of length $\lceil k/2 \rceil - 1$ each
        such that $2q_1 + 1$ and $2q_2 + 1$ are both prime.
    $p_1 \leftarrow 2q_1 + 1$; $p_2 \leftarrow 2q_2 + 1$; $n \leftarrow p_1 p_2$
    Randomly choose $\alpha \in Z_n^*$.
    $(\beta, r) \leftarrow A(n, \alpha)$
    If $1 < r \leq 2^l$ and $\beta^r \equiv \alpha \pmod{n}$ then return 1 else return 0
```

Let $\mathbf{Succ}(k, l, A) = Pr[\mathsf{Break\text{-}Strong\text{-}RSA}(k, l, A) = 1]$. Let $\mathbf{InSec}^{SRSA}(k, l, t)$ be the maximum of $\mathbf{Succ}(k, l, A)$ over all the adversaries A who run in time at most t. Our assumption is that $\mathbf{InSec}^{SRSA}(k, l, t)$, for some t polynomial in k, is negligible in k. The smaller the value of l, of course, the weaker the assumption.

In fact, for a sufficiently small l, our assumption follows from a variant of the fixed-exponent RSA assumption. Namely, assume that there exists a constant ϵ such that, for every r, the probability of computing, in time t, an r-th root of a random integer modulo a k-bit product of two safe primes, is at most 2^{-k^ϵ}. Then, $\mathbf{InSec}^{SRSA}(k, l, t) < 2^{l-k^\epsilon}$, which is negligible if $l = o(k^\epsilon)$.

3 The New Scheme

3.1 Background

Our construction relies on proofs of knowledge of discrete logarithms in groups of hidden order [FO97], which are recalled in this section.

We first recall the well-known construction due to Schnorr [Sch91] for a group of *known* order. Let p be a safe prime (i.e. $p = 2q + 1$, q - also prime) and let g be a generator of QR_p, the group of quadratic residues modulo p of size $|QR_p| = \frac{p-1}{2} = q$. Let the public key be $PK = (y, g, p)$, and the secret key be $SK = (x, g, p)$. We give PK to the verifier and SK to the prover. The following protocol allows the prover to convince the verifier that the prover knows $x \equiv \log_g y \pmod{p}$. Let l be a security parameter; the *knowledge error* (informally, the probability of prover cheating), as defined in [BG92], will be 2^{-l}.

Prover		**Verifier**
$t \overset{R}{\leftarrow} [0, q-1]$		
$d \leftarrow g^t \bmod p$		
	$\xrightarrow{d \text{ - commitment}}$	
		$\sigma \overset{R}{\leftarrow} \{0,1\}^l$
	$\xleftarrow{\sigma \text{ - challenge}}$	
$s \leftarrow t - \sigma x \bmod q$		
	$\xrightarrow{s \text{ - response}}$	
		$d \overset{?}{=} g^s y^\sigma \bmod p$

This protocol is both honest-verifier zero-knowledge and a proof of knowledge. Hence, it makes an identification scheme secure against passive attacks, and therefore can be transformed into a signature scheme secure against chosen-message attacks in the random oracle model using the Fiat-Shamir transform [FS86, PS96, AABN02]. In the resulting signature scheme, the signer acts just like the above prover, except that it computes σ as $H(M, d)$, where H is a cryptographic hash function (modeled as a random oracle), and outputs (s, σ) as a signature. The verifier, who is now non-interactive, computes $d' \leftarrow g^s y^\sigma$ and checks if $\sigma \overset{?}{=} H(M, d')$.

Although the use of the order q of the group seems crucial in the above protocol, it turns out not to be needed, as shown in [FO97] (see section 4.1 of [CM98] for a more accessible write-up). Specifically, instead of working modulo a safe prime p, we will work modulo a composite $n = p_1 p_2$, where both p_1 and p_2 are safe primes (i.e. $p_i = 2q_i + 1$, where q_i are also primes for $i = 1, 2$). It is easy to see that the group QR_n of quadratic residues modulo n has size $|QR_n| = \frac{(p_1-1)(p_2-1)}{4} = \frac{2q_1 2q_2}{4} = q_1 q_2$ and is cyclic (by Chinese Remainder Theorem, because QR_{p_1} and QR_{p_2} are both cyclic). Let g be a generator of QR_n (Chinese Remainder theorem implies that a random element of QR_n is very likely to be a generator, unless it is congruent to 1 modulo p_1 or p_2). We will now work in QR_n, instead of QR_p, and will not require the prover or the verifier to know the factorization of n or, equivalently, the order of QR_n.

As in the previous case, $y = g^x \bmod n, PK = (y, g, n)$ and $SK = (x, g, n)$. Let $k = |n|$ be length of n in bits, and $\epsilon > 1$ be another security parameter (the amount of information leaked by the signer, or, more formally, exact statistical zero-knowledgeness, will be roughly $2^{-(l+k)(\epsilon-1)}$). The reasonable parameter values are $k = 1024, l = 160, \epsilon = 1.07$ (see [CM98] for an analysis of statistical zero-knowledgeness of this protocol, from which the value for ϵ follows).

Prover		**Verifier**
$t \overset{R}{\leftarrow} \{0,1\}^{\epsilon(l+k)}$		
$d \leftarrow g^t \bmod n$		
	$\xrightarrow{d \text{ - commitment}}$	
		$\sigma \overset{R}{\leftarrow} \{0,1\}^l$
	$\xleftarrow{\sigma \text{ - challenge}}$	
$s \leftarrow t - \sigma x$		
	$\xrightarrow{s \text{ - response}}$	
		$d \overset{?}{=} g^s y^\sigma \bmod n$

Note that the main difference from the previous protocol is that s is not computed modulo the group size, but rather as an integer. Note also that the set from which t is selected somewhat larger than before, to provide statistical zero-knowledgeness.

This protocol is a secure identification scheme under the Strong-RSA assumption; hence, like the previous protocol, it can be converted to a signature scheme using the Fiat-Shamir transform.

3.2 Our Forward-Secure Scheme

Our scheme relies essentially only on the above protocol, and on the fact that squaring in QR_n is a one-way permutation. Specifically, the public key contains a modulus n as above, and a value $v \in QR_n$. The secret key for time period j contains the root $c_j \in QR_n$ of $1/v$ of degree 2^{T-j+1}. Hence, Key Update is just squaring: $c_{j+1} \leftarrow c_j^2 \bmod n$.

To sign messages in time period j, the signer applies the Fiat-Shamir transform to the interactive proof of knowledge of c_j. So all that remains is to show how to construct a proof of knowledge of 2^{T-j+1}-th root of $1/v$ modulo n.

Perhaps the first thing that comes to mind is to use something similar to the Ong-Schnorr [OS90] identification scheme that uses 2^m-th roots. However, the [OS90] construction is not actually a proof of knowledge of 2^m-th root, as pointed out in Shoup [Sho96]. Indeed, the forward-secure scheme of [AR00] based on [OS90] is quite inefficient for this very reason—because the signer needs to know a (2^{m+l})-th root, while the knowledge extractor can only obtain a 2^m-th root (see [AR00] for further explanation).

Therefore, we use a different technique, inspired by [ACJT00] and [Son01]. Let y be a generator of QR_n (included in both the public and the secret keys). The prover "blinds" c_j by a random power of y: $A \leftarrow c_j y^w \bmod n$ for a random w. Using the protocol from the previous section, the prover then proves knowledge of the discrete logarithm of $vA^{2^{T-j+1}}$ to the base $y^{2^{T-j+1}}$.

Completeness follows from the fact that prover knows w, which is the required discrete logarithm:

$$\left(y^{2^{T-j+1}}\right)^w = A^{2^{T-j+1}}/c_j^{2^{T-j+1}} = vA^{2^{T-j+1}}.$$

Proof of knowledge property follows from the fact that knowledge of such A and w is equivalent to knowledge of a root of $1/v$ of degree 2^{T-j+1}, which can simply be computed as A/y^w from the previous equation. Note that the verifier has to check that A is not 0 modulo n, because the proof for discrete logarithms is not sound in that case.

There is a minor concern: namely, because the prover does not know the order of the group QR_n, how can the prover select w so that y^w is a uniformly selected random element of QR_n? We address it by relaxing the requirement from perfect uniformity to "statistical uniformity." Namely, we use $(n-1)/4$ as a good approximation to $|QR_n| = \phi(n)/4$, and simply select w as an integer between 0 and $(n-1)/4$. Note that the chances that $w > |QR_n|$ are $\frac{n-1-\phi(n)}{n-1} < \frac{p_1+p_2}{n-1} \approx \frac{2^{k/2}+2^{k/2}}{2^k} \approx 2^{-k/2}$, and hence are negligible.

The resulting identification protocol is as follows.

Prover		**Verifier**
$w \xleftarrow{R} (0, \frac{n-1}{4}]$		
$A \leftarrow c_j y^w \bmod n$		
	$\xrightarrow{\quad A \quad}$	
$r \xleftarrow{R} \{0,1\}^{\epsilon(l+k)}$		
$d \leftarrow (y^{2^{T-j+1}})^r \bmod n$		
	$\xrightarrow{d \text{ - commitment}}$	
		$\sigma \xleftarrow{R} \{0,1\}^l$
	$\xleftarrow{\sigma \text{ - challenge}}$	
$s \leftarrow r - \sigma w$		
	$\xrightarrow{s \text{ - response}}$	
		$d \overset{?}{=} (y^{2^{T-j+1}})^s (A^{2^{T-j+1}} v)^\sigma \bmod n$
		and $A \overset{?}{\not\equiv} 0 \pmod n$

In the above protocol, the prover needs to compute A only once to blind c_j; using the same A multiple times does not hurt security of this scheme. The value $y^{2^{T-j+1}}$ also need be computed only once, and can be saved and reused by the prover. In addition, the value d can be computed off-line, before the protocol starts. Only s has to be computed on-line.

If the verifier interacts with the same prover multiple times in time period j, the verifier can also save and reuse $y^{2^{T-j+1}}$ and $A^{2^{T-j+1}}$.

The conversion from this identification scheme to the signature scheme is straightforward [FS86, AABN02], using a hash function $H : \{0,1\}^* \to \{0,1\}^l$, for a security parameter l. See Figure 1 for the resulting KREUS forward-secure signature scheme.

4 Performance

4.1 Improving Signer Costs via Pebbling

Note that the signer has to compute, in time period j, the value $\tilde{y} = y^{2^{T-j+1}} \bmod n$. In other words, the signer needs to compute, in order, the values $y_1, y_2, \ldots, y_T$, where $y_T = y$ and $y_{j-1} = y_j^2 \bmod n$. Note that this would be easy if the order of the values was reverse: the signer could just square the previous value each time. The direction we need is harder.

This is exactly the problem that Jakobsson [Jak02] and Coppersmith and Jakobsson [CJ02] address, except that they consider any one-way (hash) function, whereas we are interested specifically in modular squaring. They show that one can traverse a one-way chain by storing a few of its elements and performing a few computations per step. Specifically, the simple algorithm of [Jak02] requires $\lceil \log_2 T \rceil$ values stored and $\lceil \log_2 T \rceil$ steps in each time period; the much more complex one of [CJ02] requires about $\log_2 T + \log_2 \log_2 T$ values stored and further reduces the computation cost to about $0.5 \log_2 T$. Both algorithms use "pebbling" techniques, similar in spirit to those of [IR01].

In Figure 2, we present, in detail, a simple self-contained implementation of the algorithm of [Jak02] for use with our signature scheme. It is an adaptation to the pebbling algorithm of Jakobsson of the implementation given in [IR02]. While this description suffices to implement the algorithm, we refer the reader interested in the intuition to [Jak02].

```
**KeyGeneration**$(k, T)$
// $k$ - size of the keys in bits, $T$ - number of time periods
    Generate random $(\lceil \frac{k}{2} \rceil - 1)$-bit primes $q_1$, $q_2$, such that
            $p_1 \overset{def}{=} 2q_1 + 1$ and $p_2 \overset{def}{=} 2q_2 + 1$ are both prime
            // See [CS00] for an excellent discussion on efficient generation of safe primes
    $n \leftarrow p_1 p_2$
    $c_0 \overset{R}{\leftarrow} Z_n^*$
    $c_1 \leftarrow (c_0)^2 \bmod n$
    $v \leftarrow 1/\left(c_1^{2^T}\right) \bmod n$
    $y \overset{R}{\leftarrow} QR_n$ // One may check that $y$ is a generator of $QR(n)$, but this is not necessary,
                    since it will hold with overwhelming probability of $1 - \frac{q_1+q_2-1}{q_1 q_2}$
    $SK_1 \leftarrow (1, n, T, c_1, y)$
    $PK \leftarrow (n, T, v, y)$
    **return** $(SK_1, PK)$

**KeyUpdate**$(SK_j)$
// $SK_j$ is the secret key for time period $j = 1, 2, ..., T$
    Parse $SK_j$ as $(j, n, T, c_j, y)$
    **if** $j = T$ **then return** $nil$ // note that $nil$ here denotes the empty string
            **else** $c_{j+1} \leftarrow (c_j)^2 \bmod n$;
                **return** $SK_{j+1} \overset{def}{=} (j + 1, n, T, c_{j+1}, y)$

**Sign**$(SK_j, M, k, l, \epsilon)$
// $M$ is the message; $SK_j$ is the secret key for time period $j$; and $k, l$ and $\epsilon$ are the security parameters
        Parse $SK_j$ as $(j, n, T, c_j, y)$
Once Per Time Period:
        $w \overset{R}{\leftarrow} (0, \frac{n-1}{4}]$
        $A \leftarrow c_j y^w \bmod n$
        $\tilde{y} \leftarrow y^{2^{T-j+1}} \bmod n$
For Each Signature:
        $r \overset{R}{\leftarrow} \{0, 1\}^{\epsilon(l+k)}$
        $d \leftarrow (\tilde{y})^r \bmod n$
        $\sigma \leftarrow H(j, A, d, M)$
        $s \leftarrow r - \sigma w$
        **return** $(s, \sigma, j, A)$

**Verify**$(PK, M, (s, \sigma, j, A))$
// $M$ is the message, $PK$ is the public key, and $(s, \sigma, j, A)$ is the purported signature for time period $j$
        Parse $PK$ as $(n, T, v, y)$
Once Per Time Period Per Signer:  $\tilde{A} \leftarrow A^{2^{T-j+1}} \bmod n$; $\tilde{y} \leftarrow y^{2^{T-j+1}} \bmod n$
For Each Signature:               $d' \leftarrow (\tilde{y})^s (v\tilde{A})^\sigma \bmod n$
                                  **if** $\sigma = H(j, A, d', M)$ **and**
                                     $A \not\equiv 0 \pmod{n}$ **then return** 1 **else return** 0
```

Figure 1: Algorithms of the KREUS forward-secure signature scheme with security parameters k, l, ϵ

```
In **KeyGeneration** add:
    define data structure Pebble = $\{\tilde{y}, pos, rb, re\}$
        // $\tilde{y}$ is the value to be used in time period pos (in each pebble $\tilde{y} = y^{2^{T-pos+1}}$);
        // pos stands for position, "r" for "responsibility," "b" for "begin" and "e" for "end"
    $TR \leftarrow 1$; while $TR < T$ do $TR \leftarrow 2 \cdot TR$ // Let TR be the smallest power of 2 greater than T
    let Pebble $P \leftarrow \{y^2, T, 1, TR\}$; $L \leftarrow$ a list of Pebbles (initially, just the single element $P$)
    for $i \leftarrow -(T-4)/2$ to 1 (inclusive) do // initialize by reaching position 1
        PebbleRound($L, i$)
    remove the first element of $L$; its $\tilde{y}$ value is $y^{2^T}$ and is to be used in the first time period

In **KeyUpdate** at the end of time period $j$ add:
    PebbleRound($L, j+1$)
    remove the first element of $L$; its $\tilde{y}$ value is $y^{2^{T-j}}$ to be used in time period $j+1$

Procedure PebbleRound($L, i$)
    for each Pebble $P$ in $L$, in order, do
        Let $d = P.pos - P.rb - 2(P.rb - i)$ // Compute how much to move (it will be at most 2)
        if $d = 1$ then MoveLeft($P$) // Move once
        if $d = 2$ then MoveLeft($P$); MoveLeft($P$) // Move twice

Procedure MoveLeft($P$)
    while $P.pos \leq P.re$ do // Can't move until pos > re, or else won't be able to reach re
        // Spawn a new pebble and delegate upper half of responsibility to it
        $mid \leftarrow \lfloor (P.rb + P.re)/2 \rfloor + 1$
        if $mid \leq T$ then // The new pebble is needed only if it is for a "real" time period, T or before
            let Pebble $P1 \leftarrow (P.\tilde{y}, P.pos, mid, P.re)$; insert $P1$ into $L$ immediately following $P$
        $P.re \leftarrow mid - 1$
    $P.\tilde{y} \leftarrow P.\tilde{y}^2 \bmod n$; $P.pos \leftarrow P.pos - 1$ // this is the actual move left
```

Figure 2: *Pebbling techniques to speed up computations of $\tilde{y}$*

Thus, in our scheme, much like in the scheme of [IR01], a time/memory tradeoff is possible: if the signer stores $\log_2 T$ values, then the cost per time period goes down from linear in T to logarithmic in T. Note, however, that both in our scheme and in [IR01], if the signer does not use pebbling, then the linear in T cost needs to be incurred only in those time periods in which signatures are issued; whereas if pebbling is used, then the logarithmic in T cost needs to be incurred in *every* time period in order for the pebbling algorithm to work correctly.

4.2 Performance Comparison

In Tables 1 and 2 we compare the performance of our scheme to the performance of other Fiat-Shamir-like forward-secure signature schemes of [BM99], [AR00] and [IR01]. The comparison is in

	Update	Off-line Signing per time period	Off-line Signing per message	On-line Signing
[BM99]	l	n/a	$T-j$	$l/2$
[AR00]	l	n/a	$l(T-j)$	$3l/2$
[IR01]	$3l/2$	$3l(T-j)/2$	$3l/2$	$3l/2$
with $k \log_2 T$ storage	$(3l/2)\log_2 T$	0	$3l/2$	$3l/2$
KREUS	1	$T-j+3k/2$	$3\epsilon(l+k)/2$	1
with $k \log_2 T$ storage	$\log_2 T$	$3k/2$	$3\epsilon(l+k)/2$	1

Table 1: Signer costs (in modular multiplications) of various forward-secure schemes

	Once per signer per time period	For every signature
[BM99]	n/a	$T-j+l/2$
[AR00]	n/a	$l(T-j)+3l/2$
[IR01]	n/a	$3l$
KREUS	$2(T-j)$	$3(\epsilon(l+k)+l)/2$

Table 2: Verifier costs (in modular multiplications) of various forward-secure schemes

terms of the number of modular multiplications required, expressed as functions of the total number of time periods T, the key size k, the hash length l, security parameter ϵ, and a current time period j. We assume that a modular exponentiation with an l-bit exponent is roughly equal to $3l/2$ modular multiplications; to improve readability, we ignore small additive constants. The "generic" schemes of [BM99], [Kra00] and [MMM02] are not included in our comparison because of their fundamentally different design (their performance depends on the ordinary signature scheme with which they are instantiated).

Table 1 contains three metrics corresponding to Key Update, off-line Signing and on-line Signing parts of each scheme. Off-line Signing is additionally subdivided into two components: off-line Signing per time period and off-line Signing per message. By "off-line Signing per time period" we mean the computations that are needed only once per time period, and only in those time periods in which signatures are issued. This is different from Key Update, which needs to be performed in every time period, whether or not signatures are issued. This distinction—between off-line Signing per time period and Key Update—has not been made before. In particular, while the authors of [IR01] state that their scheme has a high update cost, in our terms it has a high off-line signing per time period cost. This distinction is important: if time periods are short for added security, then it is quite possible that no signatures will be issued in many of the time periods (and hence much of the computation will not be necessary).

Table 2 presents verifier's costs, subdivided into the costs the verifier incurs only once per time period (for a given signer), and the costs for each signature.

We evaluate the expressions in Table 1 for specific practical parameter values in Tables 3 and 4. Table 3 considers $T = 512$ (more than one update per day for a year), and Table 4 considers $T = 2^{25}$ (more than one update per second for a year).

Finally, these tables allows us to determine when our scheme has lower total signer costs than [IR01] for the above parameter values of k, l, ϵ, both with and without the extra storage. We obtain the following results: without the extra storage, our scheme has lower total signer costs if no more

	Update	**Off-line Signing per**		**On-line**
		time period	**message**	**Signing**
[BM99]	160	n/a	256	80
[AR00]	160	n/a	$40,960$	240
[IR01]	240	$61,440$	240	240
with $1,152$ bytes storage	$2,160$	0	240	240
KREUS	1	$1,792$	$1,901$	1
with $1,152$ bytes storage	9	$1,536$	$1,901$	1

Table 3: Performance of different schemes (in modular multiplications) with $T = 512, k = 1024, l = 160, j = T/2, \epsilon = 1.07$

	Update	**Off-line Signing per**		**On-line**
		time period	**message**	**Signing**
[BM99]	160	n/a	16×10^6	80
[AR00]	160	n/a	2.6×10^9	240
[IR01]	240	4×10^9	240	240
with $3,200$ bytes storage	$6,000$	0	240	240
KREUS	1	16×10^6	$1,901$	1
with $3,200$ bytes storage	25	$1,536$	$1,901$	1

Table 4: Performance of different schemes (in modular multiplications) with $T = 2^{25}, k = 1024, l = 160, j = T/2, \epsilon = 1.07$

that 42 signatures per time period are issued for $T = 512$, and if no more that 2.8 million signatures per time period are issued for $T = 2^{25}$. With the extra storage, the relative performance of [IR01] improves, and our scheme's total signer costs are lower if no more than 0.7 signatures per time period are issued for $T = 512$, and no more than 3 signatures per time period are issued for $T = 2^{25}$.

5 Security

The security proof is fairly standard: one can use the forking lemma technique of [PS96] and/or the reduction from identification to signatures of [AABN02]. We omit it here for lack of space; we do, however, point out its most salient features.

The security proof boils down to running the forger and simulating answers to the forger's signature queries. For this simulation, one relies on the statistical zero-knowledgeness of the identification scheme, using techniques similar to the ones of [CM98]. After a successful simulation, one obtains from the forger two signatures (s, σ, j, A) and (s', σ', j, A) for the same d. In other words,

$$\left(y^{2^{T-j+1}}\right)^{s} \left(vA^{2^{T-j+1}}\right)^{\sigma} \equiv \left(y^{2^{T-j+1}}\right)^{s'} \left(vA^{2^{T-j+1}}\right)^{\sigma'} \pmod{n}.$$

Letting $\tilde{s} = s - s'$ and $\tilde{\sigma} = \sigma' - \sigma$, we obtain the equation

$$\left(y^{2^{T-j+1}}\right)^{\tilde{s}} \equiv \left(vA^{2^{T-j+1}}\right)^{\tilde{\sigma}}.$$

We now consider two types of forgers. In the first type, $\tilde{\sigma}$ divides $\tilde{s}$. To violate the Strong-RSA assumption using such forger, the reduction, on input a random $\alpha \in Z_n^*$, sets $v = \alpha^{2^{T-j+1}}$ (this will enable the reduction to answer breakin query after time period j). Then the above equation becomes $\left(y^{\tilde{s}/\tilde{\sigma}}/A\right)^{2^{T-j+1}} \equiv \alpha^{2^{T-j+1}}$. Because squaring is a permutation over QR_n (because n is a product of two safe primes, and hence a Blum integer), this gives that $\left(y^{\tilde{s}/\tilde{\sigma}}/A\right)^2 \equiv \alpha^2$. In other words, the reduction is able to compute a square root of α^2; because α is a random element of Z_n^*, this is equivalent to factoring n with probability 1/2, and hence, in particular, violates the Strong-RSA assumption.

In the second type of forger, $\tilde{\sigma}$ does not divide $\tilde{s}$. To violate the Strong-RSA assumption using such forger, the reduction, on input a random $\alpha \in Z_n^*$, sets $y = \alpha^2$, picks $c_0 \in Z_n^*$, lets $c_1 = c_0^2$ and $v = 1/\left(c_1^{2^T}\right)$, just like the true signer. Then the above equation becomes $y^{\tilde{s}2^{T-j+1}} \equiv (A/c_j)^{\tilde{\sigma}2^{T-j+1}}$. By the so-called "Shamir's trick" [Sha83], this equation enables us to obtain a root of y of degree $r = \tilde{\sigma}2^{T-j+1}/\gcd\left(\tilde{\sigma}2^{T-j+1}, \tilde{s}2^{T-j+1}\right) = \tilde{\sigma}/\gcd(\sigma, s) > 1$. If r is even, this gives us another square root of α^2, thus again allowing us to factor n with probability 1/2. If r is odd, this gives us a root of α of degree $r > 1$, thus breaking the Strong-RSA assumption (note also that $r < 2^l$, because σ and σ' are at most l bits long, because they are output by H).

Acknowledgments

We thank Jan Camenisch and Markus Jakobsson for explaining to us the history of building blocks of our scheme.

References

[AABN02] Jee Hea An, Michel Abdalla, Mihir Bellare, and Chanathip Namprempre. From identification to signatures via the Fiat-Samir transform: Minimizing assumptions for security and forward-security. In Knudsen [Knu02].

[ACJT00] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In Mihir Bellare, editor, *Advances in Cryptology—CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 255–270. Springer-Verlag, 20–24 August 2000.

[And97] Ross Anderson. Invited lecture. In *Fourth Annual Conference on Computer and Communications Security*. ACM, 1997. Summary appears in [And01].

[And01] Ross Anderson. Two remarks on public key cryptology. `http://www.cl.cam.ac.uk/users/rja14/`, 2001.

[AR00] Michel Abdalla and Leonid Reyzin. A new forward-secure digital signature scheme. In Tatsuaki Okamoto, editor, *Advances in Cryptology—ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 116–129, Kyoto, Japan, 3–7 December 2000. Springer-Verlag. Full version available from the Cryptology ePrint Archive, record 2000/002, `http://eprint.iacr.org/`.

[BG92] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In Ernest F. Brickell, editor, *Advances in Cryptology—CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 390–420. Springer-Verlag, 1993, 16–20 August 1992.

[BM99] Mihir Bellare and Sara Miner. A forward-secure digital signature scheme. In Michael Wiener, editor, *Advances in Cryptology—CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 431–448. Springer-Verlag, 15–19 August 1999. Revised version is available from `http://www.cs.ucsd.edu/~mihir/`.

[BP97] Niko Barić and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In Walter Fumy, editor, *Advances in Cryptology—EUROCRYPT 97*, volume 1233 of *Lecture Notes in Computer Science*, pages 480–494. Springer-Verlag, 11–15 May 1997.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communication Security*, pages 62–73, November 1993. Revised version available from `http://www.cs.ucsd.edu/~mihir/`.

[CJ02] Don Coppersmith and Markus Jakobsson. Almost optimal hash sequence traversal. In *6th International Financial Cryptography Conference*, March 11–16 2002.

[CM98] Jan Camenisch and Markus Michels. A group signature scheme based on an RSA-variant. Technical Report RS-98-27, BRICS, University of Aarhus, November 1998.

[CS00] Ronald Cramer and Victor Shoup. Signature schemes based on the strong RSA assumption. *ACM Transactions on Information and System Security*, 3(3):161–185, 2000.

[FO97] Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In Burton S. Kaliski Jr., editor, *Advances in Cryptology—CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 16–30. Springer-Verlag, 17–21 August 1997.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology—CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer-Verlag, 1987, 11–15 August 1986.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[GQ88] Louis Claude Guillou and Jean-Jacques Quisquater. A "paradoxical" indentity-based signature scheme resulting from zero-knowledge. In Shafi Goldwasser, editor, *Advances in Cryptology—CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*, pages 216–231. Springer-Verlag, 1990, 21–25 August 1988.

[IR01] Gene Itkis and Leonid Reyzin. Forward-secure signatures with optimal signing and verifying. In Joe Kilian, editor, *Advances in Cryptology—CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 332–354. Springer-Verlag, 19–23 August 2001.

[IR02] Gene Itkis and Leonid Reyzin. Forward-secure signatures with optimal signing and verifying. *Cryptobytes*, 5(2), 2002.

[Jak02] Markus Jakobsson. Fractal hash sequence representation and traversal. In *2002 IEEE International Symposium on Information Theory*, June 30–July 5 2002.

[Knu02] Lars Knudsen, editor. *Advances in Cryptology—EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*. Springer-Verlag, 28 April–2 May 2002.

[Kra00] Hugo Krawczyk. Simple forward-secure signatures from any signature scheme. In *Seventh ACM Conference on Computer and Communication Security*. ACM, November 1–4 2000.

[Mau96] Ueli Maurer, editor. *Advances in Cryptology—EUROCRYPT 96*, volume 1070 of *Lecture Notes in Computer Science*. Springer-Verlag, 12–16 May 1996.

[MMM02] Tal Malkin, Daniele Micciancio, and Sara Miner. Efficient generic forward-secure signatures with an unbounded number of time periods. In Knudsen [Knu02].

[OS90] Heidroon Ong and Claus P. Schnorr. Fast signature generation with a Fiat Shamir-like scheme. In I. B. Damgård, editor, *Advances in Cryptology—EUROCRYPT 90*, volume 473 of *Lecture Notes in Computer Science*, pages 432–440. Springer-Verlag, 1991, 21–24 May 1990.

[PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Maurer [Mau96], pages 387–398.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[Sha83] Adi Shamir. On the generation of cryptographically strong pseudorandom sequences. *ACM Transactions on Computer Systems*, 1(1):38–44, February 1983.

[Sho96] Victor Shoup. On the security of a practical identification scheme. In Maurer [Mau96], pages 344–353.

[Son01] Dawn Xiaodong Song. Practical forward secure group signature schemes. In *Eighth ACM Conference on Computer and Communication Security*, pages 225–234. ACM, November 5–8 2001.