Forward-Secure Digital Signature Schemes with Optimal Computation and Storage of Signers

Jihye Kim[1] and Hyunok Oh[2(✉)]

[1] Kookmin University, Seoul, Korea
jihyek@kookmin.ac.kr
[2] Hanyang University, Seoul, Korea
hoh@hanyang.ac.kr

Abstract. Forward-secure signatures minimize damage by preventing forgeries for past time periods when a secret key is compromised. Forward-secure signature schemes are useful for various devices such as logging systems, unattended sensors, CCTV, dash camera, etc. Considering sensors equipped with limited resources and embedded real-time systems with timing constraints, it is necessary to design a forward-secure signature scheme with minimal overhead on signer's side.

This paper proposes the first forward secure digital signature schemes with constant complexities in signature generation, key update, the size of keys, and the size of a signature. The proposed algorithms have $O(k^3)$-time complexity for each signing and key update algorithm and $O(k)$-size secret keys where k is an RSA security parameter. We prove the security of our proposed schemes under the factoring assumption in the random oracle model and present a concrete implementation of our schemes to demonstrate their practical feasibility.

Keywords: Forward secure · Digital signature · Fast signing/update · Factoring

1 Introduction

Forward-secure signature schemes mitigate the damage caused by a secret key exposure. The role of the digital logs and data as forensic values has boosted the need for strong authenticity of data. For example, audit logs record the "what happened when by whom" of the system. The forensic value of audit logs makes them an attractive target for attackers [1]. An active attacker compromising a logging machine can modify log entries related to the past, erasing records of the attacker's previous break-in attempts. Forward secure digital signature schemes, of which goal is to preserve the validity of past signatures even if the current secret key has been compromised, become an effective solution to prevent this active attack as well as to provide strong authenticity for the recorded video frames.

S. De Capitani di Vimercati and F. Martinelli (Eds.): SEC 2017, IFIP AICT 502, pp. 523–537, 2017.
DOI: 10.1007/978-3-319-58469-0_35

Briefly, a forward-secure signature scheme divides the total time into T time periods and uses a different secret key in each time period (while the public key remains fixed). Each subsequent secret key is computed from the current secret key via a key update algorithm. Although it is ideal to have constant complexities regardless of the parameter T in computations and storage sizes overall, it is a challenging work. In the first forward-secure signature scheme proposed by Anderson [2], the size of secret key increases linearly with T. In the Bellare and Miner (BM) scheme [3] both public and secret key sizes are constant, but the signing and the verification time (of a single signature) grows linearly with T. Itkis and Reyzin [4] (IR) propose a scheme to have constant complexities in signing and verification, however, at the expense of key update time and the secret key size which grow logarithmically with T. Malkin et al. [5] (MMM) proposed a generic forward secure signature scheme based on a hash chain tree with a constant size public key. Although the secret key size, the signature size, signing and verifying time are $O(\log T)$, theoretically, the actual computation time and the storage requirement seem to be independent of T since the hash computation and size are relatively small compared with a public key signature scheme which it uses internally. However a signature in MMM contains two public keys and two signatures with an $O(\log T)$ size hash chain. The resulting signature size is 4 times larger than BM and IR.

One of main hurdles when forward-secure signature schemes are deployed in the real systems is located in their non-constant signing/update overhead. For example, in the video recording devices for streaming applications, a captured video frame is compressed periodically, e.g., every 33 ms. If a signature is generated for each frame, the signature computation with key update should be completed within 33 ms. When an incident occurs, the stored video frames and their signatures are used for forensic analysis. In this scenario, the signing and update time should be short enough to meet the time constraint. On the contrary, the verification of the signatures may be performed when a forensic is required.

The goal of this paper is to construct forward-secure signature schemes efficient enough to cover even resource- and/or time-constraint devices such as unattended sensors and surveillance real-time streaming systems. To achieve this goal, the computation and size complexities on signer's side should be short and constant at least. For the practical usage like other previous schemes, the public key size also needs to be constant. Setup and verification times not directly related with the signing device are comparatively less important.

Contributions. Our schemes extend the Bellare-Miner (BM) scheme [3] and the Abdalla-Reyzin (AR) scheme [6] to provide *a short and constant signature computation time*. The proposed schemes, denoted as Fast-BM and Fast-AR, require the same constant size memory for secret/public keys, and generate the same constant size signatures as BM and AR, respectively. The signature computation time complexity is $O(k^3)$ in our schemes, while the signature computation time complexity is $O(k^2T)$ in the BM scheme, and $O(k^2lT)$ in the AR scheme, where l is a security parameter representing the bit length of the hash output,

k is a security parameter denoting the bit length in RSA (modulo N is k-bit integer), and T represents the number of periods. Surprisingly, there is no significant performance degradation in other metrics, while we optimize the signing algorithms of BM and AR. In the experiment, our algorithms generate a signature and update their secret keys in 25 ms with security parameters $k = 2048$ and $l = 160$ regardless of the total number of periods T. The results show that our proposals are fast enough not only for normal applications but also for real-time streaming applications. The proposed Fast-BM and Fast-AR schemes are secure under the factoring assumption in the random oracle model.

We begin, in the next section, by describing background for forward secure digital signature schemes. Section 3 proposes our fast forward secure digital signature schemes with explaining the underlying schemes. Section 4 discusses the security of the proposed schemes. In Sect. 5, experimental results present quantitative measurements. We describe related work in Sect. 6 and summarize our conclusion in Sect. 7.

2 Background

This section reviews the syntax and security definitions of a forward secure digital signature scheme and defines its formal notion of security. All definitions provided here are based on those given in [3,6]. We also present the underlying cryptographic assumptions that our proposal relies on. We introduce some basic notations. If S is a set then $s \xleftarrow{\$} S$ denotes the operation of picking a random element s of S. We write $A(x, y, \cdots)$ to indicate that A is an algorithm with inputs $x, y, \cdots$ and by $z \leftarrow A(x, y, \cdots)$ we denote the operation of running A with inputs $(x, y, \cdots)$ and letting z be the output.

2.1 Forward Secure Signature Schemes

A forward secure signature scheme is a key-evolving signature scheme. We follow the definition of forward secure signature schemes in [3,6].

Definition 1 (Key-evolving signature scheme). *A key-evolving digital signature scheme is a set of four algorithms:* $\mathsf{FSIG} = (\mathsf{Setup}, \mathsf{Sig}, \mathsf{Upd}, \mathsf{Ver})$, *where:*

- Setup: The key generation algorithm is a probability algorithm which takes as input a security parameter l and the total number of periods T and returns a pair (SK_0, PK), the initial secret key and the public key.
- Sig: The signing algorithm takes as input the secret key SK_i for the current time period i and the message M to be signed, and returns a pair $\langle i, s \rangle$, the signature of M for time period i.
- Upd: The key update algorithm takes as input the secret key SK_i for the current interval and returns a new secret key SK_{i+1} for the next interval.
- Ver: The verification algorithm takes as input the public key PK, message M and a candidate signature $\langle i, s \rangle$, and returns 1 if $\langle i, s \rangle$ is a *valid* signature of M, or 0, otherwise. It is required that $\mathsf{Ver}_{\mathsf{PK}}(\mathsf{M}, \mathsf{Sig}_{\mathsf{SK_i}}(\mathsf{M})) = 1$ for every message M and time period i.

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

Security: The adversary executes the usual adaptive chosen-message attack (cma) until it breaks in and learns the secret key for a given time period. The adversary is then considered successful if it can create a valid forgery on a new message for a *previous* time period. Formally, this adversary, denoted by F, is modeled via the following experiment. The adversary, denoted by F, runs in three phases. In the cma phase, F has access to a sign oracle. F is allowed to query multiple signatures in the same period. In the break-in phase, F is given the secret key SK_j for the specific interval j. Finally, in the forgery phase (forge), F outputs a pair of a signature and a message. The adversary is successful if it forges a signature of any new message (not previously queried to the signing oracle) for any time period prior to j. The formal experiment is described in the following:

F-Forge$(\mathsf{FSIG}, \mathsf{F})$:

$(SK_0, PK) \xleftarrow{\$} \mathsf{Setup}(\mathsf{k}, \cdots, \mathsf{T})$;
$j \leftarrow 0$
repeat
  $j \leftarrow j + 1$; $SK_j \leftarrow \mathsf{Upd}(SK_j)$; $d \leftarrow F^{\mathsf{Sig}_{SK_j}(\cdot)}(\mathsf{cma}, PK)$
until $(d = \mathtt{breakin})$ or $(j = T)$
`If` $(d \neq \mathtt{breakin})$ and $(j = T)$ then $j \leftarrow T + 1$
$(M, \langle b, s\rangle) \leftarrow F(\mathtt{forge}, SK_j)$
`If` $\mathsf{Ver}(\mathsf{M}, \langle \mathsf{b}, \mathsf{s}\rangle) = 1$ and $1 \leq b < j$
  and M was not queried of $\mathsf{Sig}_{SK_b}(\cdot)$ in period b
  then return 1 else return 0

Definition 2 (Forward-security). *Let* $\mathsf{FSIG} = (\mathsf{Setup}, \mathsf{Sig}, \mathsf{Upd}, \mathsf{Ver})$ *be a key-evolving signature scheme and* F *an adversary as described above. Let* $\mathbf{Succ}^{\mathrm{fwsig}}(\mathsf{FSIG}[\mathsf{k}, \cdots, \mathsf{T}], \mathsf{F})$ *denote the probability that the experiment* $\mathbf{F\text{-}Forge}(\mathsf{FSIG}[k, \cdots, T], F)$ *returns 1. Then the insecurity of* FSIG *is the function*

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

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

3 Fast Forward Secure Digital Signature Schemes

Our proposed schemes extend the previous forward secure signature schemes proposed by Bellare and Miner (BM) [3] and by Abdalla and Reyzin (AR) [6].

Since the way of extension is the same for each, we describe our scheme focusing on the AR version (which has a simpler parameter setting). We first overview the scheme proposed by Abdalla-Reyzin (AR) in [6] and then describe our proposed schemes.

3.1 Overview of the AR Scheme

The AR scheme [6] defines T the maximum number of periods and extends the 2^l-th root signature scheme [7] to have the forward security property. The initial secret key S_0 is arranged as $2^{l(T+1)}$-th root of a public key U. For each period, the secret key is updated by raising it to the 2^l power and thus the secret key at the period j becomes $S_j = S_0^{2^{l(T+1-j)}}$. At period j, the signer proves the knowledge of the $2^{l(T+1-j)}$-th root of U, of which computational cost is proportional to T. Thus, as T increases, the signing time increases. The size of T depends on the application and is possibly large in general to avoid frequent setups. For instance, assume that a signature is generated and a secret key is updated every second. In order to provide a forward security in this device for a year, T should be no less than 31,536,000 ($=60 \times 60 \times 24 \times 365$). AR (of which signature computation depends on T) may be impractical to be used for applications with this large T setting.

3.2 Fast-AR

Algorithm 1 summarizes the key setup, the secret key update, the sign, and the verification of our proposed algorithm called Fast-AR. In the proposed algorithm, all numbers including secret keys (S_j), a public key (U), random numbers (R), and their exponentiations (Y) are chosen in $\langle g \rangle$ where g is a generator of a large subgroup in Z_N^*. In the following, we describe our approach and details of each algorithm.

Setup: We generate a safe RSA (or a safe-prime product RSA) modulus $N = pq$ where p, q, p', and q' are primes such that $p = 2p' + 1$, and $q = 2q' + 1$. Note that p and q are congruent 3 mod 4 and N becomes a Blum integer. Pick a random element g s.t. g generates a maximum subgroup in Z_N^*, i.e. $ord(g) = 2p'q'$ and s.t. $-1 \notin \langle g \rangle$. Note that this holds for about half of the elements in Z_N^*, and it is easily tested. In addition, the Jacobi symbol of g, $(g|N) = -1$.

Additionally, we compute $X = g^{2^{l(T+1)}} \bmod N$. Using g and X, we can efficiently compute $l(T+1)$ squaring operations of any group element in $\langle g \rangle$, given the order of the element. Thus, to compute $U = S_0^{2^{l(T+1)}} \bmod N$ where $S_0 = g^s \bmod N$ for some known s, we compute $X^s \bmod N$ instead. (The same technique is used in the sign algorithm.)

Since the group size $\phi(N)(= (p-1)(q-1))$ is known at setup, computation of $X = g^{2^{l(T+1)}} \bmod N$ has $O(k^3)$-time complexity. A secret key S is chosen from $\langle g \rangle$ by selecting a random number s in $Z_{N/2}$ and computing $S = g^s \bmod N$. A public key U ($\equiv S^{2^{l(T+1)}} \equiv g^{s2^{l(T+1)}} \equiv X^s (\bmod N)$) is computed by raising X to

the s as we describe above. After computing a secret key and a public key, the primes p, q and the chosen random number s are discarded. g and X are stored in a secret key.

Sign and Update: In AR, the signing overhead occurs mainly due the computation of $Y \leftarrow R^{2^{l(T+1-j)}} \bmod N$ for a chosen random R at period j. Since the computation requires $l(T+1-j)$ squaring operations of R, the computation complexity is proportional to T. Recall that we select a generator g (of a maximum subgroup of Z_N^*) and compute $g^{2^{l(T+1)}}$ denoted as X in advance. In the signing procedure of our scheme, we generate R by raising g to a random number e in $Z_{N/2}$. Then since Y is $R^{2^{l(T+1)}} \equiv g^{e2^{l(T+1)}} \equiv X^e (\bmod N)$, Y can be computed by raising X to the e. So the computation time of Y does not rely on T. After computing R and Y, e is erased. Note that the computation of Y in our scheme is different from AR: $Y = R^{2^{l(T+1)}}$ in our scheme, while $Y = R^{2^{l(T+1-j)}}$ in AR. I.e., the computation of Y in our scheme is independent of period j, unlike AR. Therefore, after we compute $X = g^{2^{l(T+1)}}$ once (in setup), it can be reused at every period. The verification in our scheme is modified accordingly, which is slightly different from AR and slower than AR, but has the same computation complexity as AR.

Algorithm 1. Fast-AR

function Setup(k, l, T)
Pick random p' and q' such that $p(=2p'+1)$, $q(=2q'+1)$, p', q' are prime, and p and q are $k/2$ bit and set $N \leftarrow pq$
Pick a random element g s.t. g generates a maximum subgroup in Z_N^*, i.e. $ord(g) = 2p'q'$, and such that $-1 \notin \langle g \rangle$
$x \leftarrow 2^{l(T+1)} \bmod (p-1)(q-1)$; $X \leftarrow g^x \bmod N$
$s \xleftarrow{\$} Z_{N/2}$; $S_0 \leftarrow g^s \bmod N$; $U \leftarrow X^s \bmod N$
$SK \leftarrow (N, T, \mathrm{g}, \mathrm{X}, 0, S_0)$; $PK \leftarrow (N, U, T)$
return (PK, SK)
end function

---

function Upd(SK) : parse SK as $(N, T, \mathrm{g}, \mathrm{X}, j, S_j)$
if j=T **then** $SK \leftarrow \epsilon$
else $SK \leftarrow (N, T, \mathrm{g}, \mathrm{X}, j+1, S_j^{2^l} \bmod N)$
end if
return SK
end function

---

function Sig(M, SK) : parse SK as $(N, T, \mathrm{g}, \mathrm{X}, j, S_j)$
$e \xleftarrow{\$} Z_{N/2}$; $R \leftarrow g^e \bmod N$; $Y \leftarrow X^e \bmod N$;$\sigma \leftarrow H(j, Y, M)$; $Z \leftarrow RS_j^{\sigma} \bmod N$
return $(j, (Z, \sigma))$
end function

---

function Ver$(M, PK, sign)$: parse PK as (N, U, T); parse $sign$ as $(j, (Z, \sigma))$
if $Z \equiv 0 (\bmod N)$ **then return** 0
else
$Y' \leftarrow Z^{2^{l(T+1)}} / U^{\sigma 2^{lj}} \bmod N$
if $\sigma = H(j, Y', M)$ **then return** 1
else return 0
end if
end if
end function

The key update algorithm is the same as AR, which requires l times squaring operations only.

Verification: As mentioned early, our verification algorithm is slightly different from AR due to the different exponentiation number for Y. While the verification tests whether Y is equal to $Z^{2^{l(T+1-j)}}U^{\sigma}$ in AR, it checks whether Y is equal to $Z^{2^{l(T+1)}}/U^{\sigma 2^{lj}}$ in our scheme. On average, our verification requires twice computation than AR.

Correctness: For a given signature of $(j,(Z,\sigma))$ for message M, the verification is to check whether $\sigma = H(j,Y',M)$ where $Y' \leftarrow Z^{2^{l(T+1)}}/U^{\sigma 2^{lj}}$. Since $Z = RS_j^{\sigma} = g^e S_0^{2^{lj}\sigma}$ and $(g^e)^{2^{l(T+1)}} = X^e = Y$, $Z^{2^{l(T+1)}} = YS_0^{\sigma 2^{lj} 2^{l(T+1)}} = YU^{\sigma 2^{lj}}$. So the verification works correctly.

4 Security Analysis

Since the proposed Fast-AR scheme is similar to the existing AR scheme except that numbers are chosen in $\langle g \rangle$ rather than Z_N^* in a signature generation, the security proof is similar to the proof of AR.

Let k and l be two security parameters. Let $p = 2p' + 1$, $q = 2q' + 1$, p', and q' be primes and $N = pq$ be a k-bit integer (Since $p \equiv q \equiv 3 \pmod 4$, N is a Blum integer). Let Q denote the set of non-zero quadratic residues modulo N. Note that for $x \in Q$, exactly one of its four square roots is also in Q. In the following description, $x \stackrel{\$}{\leftarrow} \langle g \rangle$ denotes that $r \stackrel{\$}{\leftarrow} Z_N$ and $x \leftarrow g^r \bmod N$ for $ord(g) = 2p'q'$.

Lemma 1. *Given $\alpha \neq 0$, $\lambda > 0$, $v \in Q$ and $X \in \langle g \rangle$ such that $v^{\alpha} \equiv X^{2^{\lambda}} (\mathrm{mod}\ N)$ and $\alpha < 2^{\lambda}$, one can easily compute y such that $v \equiv y^2 (\mathrm{mod} N)$.*

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

Theorem 1. *If there exists a forger $\mathcal{F}$ for $\mathsf{FSIG}[\mathsf{k},\mathsf{l},\mathsf{T}]$ that runs in time at most t, asking at most q_H hash queries and q_S signing queries, such that* $\mathbf{Succ}^{\mathrm{fwsig}}(\mathsf{FSIG}[\mathsf{k},\mathsf{l},\mathsf{T}],\mathsf{F}) \geq \epsilon$, *then there exists an algorithm $\mathcal{A}$ that factors Blum integers generated by $\mathsf{FSIG.key}(\mathsf{l},\mathsf{T})$ in expected time at most t' with probability at least ϵ', where $t' = 2t + O(k^2 lT + k^3)$, and $\epsilon' = \frac{(\epsilon - 2^{3-k} q_S (q_H+1))^2}{2T^2(q_H+1)} - \frac{\epsilon - 2^{3-k} q_S (q_H+1)}{2^{l+1}T}$.*

Proof. Suppose that there exists a forger $\mathcal{F}$ against $Fast-AR$ scheme that succeeds with ϵ in time t. We construct an algorithm $\mathcal{A}$ using $\mathcal{F}$ as a subroutine to factor a given Blum-Williams integer N with a probability of ϵ' within t' time. The goal is to find a pair (p,q) such that $N = pq$.

$\mathcal{A}$ is constructed as follows:

Setup

1. select $g \stackrel{\$}{\leftarrow} Z_N^*$ such that $-1 \notin \langle g \rangle$ [Given a safe RSA modulus N, with probability about 1/2 we have $ord(g)=2p'q'$.]
2. $w \stackrel{\$}{\leftarrow} \langle g \rangle$; $v \leftarrow w^2 \bmod N$ [Will try to find a square root of v that is different from w.]
3. $b' \stackrel{\$}{\leftarrow} \{1, \cdots, T\}$ [Choose break-in period hoping that the break-in will occur after b' period, and the forgery will be at b' or earlier.]
4. $U \leftarrow v^{2^{l(T-b')}}$ [The intention is that ${S_{b'}}^{2^l} = v \bmod N$.]
5. $PK \leftarrow (N, T, U)$ [Build a public key.]

Now, we will explain how $\mathcal{A}$ simulates.

Simulation

1. **Hash query** simulation: If $H(j, Y, M)$ is undefined then $H(j, Y, M) \stackrel{\$}{\leftarrow} \{0,1\}^l$. It returns $H(j, Y, M)$.
2. **Sign query** for M at the j-th period: If $j \leq b'$ then $Z \stackrel{\$}{\leftarrow} \langle g \rangle$; $\sigma \stackrel{\$}{\leftarrow} Z_{N/2}$; $Y \leftarrow \frac{Z^{2^{l(T+1)}}}{v^{2^{l(T-b'+j)}\sigma}} \bmod N$. Y satisfies that $Z^{2^{l(T+1)}} \equiv YU^{2^{lj}\sigma} (\mathrm{mod} N)$ since $U^\sigma = v^{2^{l(T-b')}\sigma} \bmod N$. If $H(j, Y, M)$ is defined then $\mathcal{A}$ sets $fail_1 \leftarrow \mathsf{true}$ and aborts the execution of $\mathcal{F}$; otherwise $H(j, Y, M) \leftarrow \sigma$. $\mathcal{A}$ returns $(j, (Z, \sigma))$. Since there are at most q_H entries defined in tables H, the probability that $fail_1$ happens is at most $q_H/2^l$ per sign query. Consider $j > b'$. Since secret $S_j = v^{2^{l(j-b'-1)}}$, $\mathsf{Sig}(M, SK)$ can be performed and $(j, (Z, \sigma))$ is returned.
3. **Update simulation**: If $j \leq b'$ then nothing is performed. Otherwise $\mathsf{Upd}(M, SK)$ is called.
4. **Break-in** simulation at the b-th period: If $b \leq b'$ then $\mathcal{A}$ sets $fail_3 \leftarrow \mathsf{true}$ and aborts the execution of $\mathcal{F}$. Otherwise, $\mathcal{A}$ returns S_b such that $S_b = S_{b'}^{2^{l(b-b')}} = v^{2^{l(b-b'-1)}}$.

Factoring of N

Assume that $\mathcal{F}$ outputs a forged signature $(j, (Z, \sigma))$ for a message M where σ is a hash query for $H(j, Y, M)$. Assume that the forgery period j is no later than the break-in period b' or $j \leq b'$. If $\mathcal{F}$ forges the signature without querying on $H(j, Y, M)$ then A sets $fail_2 \leftarrow \mathsf{true}$ and aborts the execution of F. $\mathcal{A}$ resets $\mathcal{F}$ with the same random tape as the first time, and runs it again, giving the exact same answers to all $\mathcal{F}$'s queries before the hash query of $H(j, Y, M)$. On the query of $H(j, Y, M)$, $\mathcal{A}$ comes up with a new answer $\sigma' \stackrel{\$}{\leftarrow} \{0,1\}^l$, sets $H(j, Y, M) \leftarrow \sigma'$. Then $\mathcal{F}$ returns $(j, (Z', \sigma'))$. If the second forgery was not based on hash query on $H(j, Y, M)$ then $\mathcal{A}$ fails.

We know the following two equations must hold: $Z^{2^{l(T+1)}} \equiv YU^{2^{lj}\sigma} \pmod N$ and $Z'^{2^{l(T+1)}} \equiv YU^{2^{lj}\sigma'} \pmod N$. Dividing, we get $(\frac{Z}{Z'})^{2^{l(T+1)}} \equiv U^{2^{lj}(\sigma-\sigma')} \pmod N$. From the setup, we know that $U \equiv v^{2^{l(T-b')}} \pmod N$. So we can write $(\frac{Z}{Z'})^{2^{l(T+1)}} \equiv v^{2^{l(T+j-b')}(\sigma-\sigma')} \pmod N$. Taking roots of degrees $2^{l(T+j-b')}$ of both sides, which we are allowed to do because both sides are in Q and remain in Q, because v is a square, $v^{\sigma-\sigma'} \equiv (\frac{Z}{Z'})^{2^{l(b'+1-j)}} \pmod N$. By applying Lemma 1, our algorithm can easily compute a square root of v, denoted as x, by setting $\alpha = \sigma - \sigma'$, $X = Z/Z'$, and $\lambda = l(b' + 1 - j)$. If $x \equiv \pm w \bmod N$ then abort. Otherwise, we compute $h \leftarrow \gcd(w - x, N)$ which is a non-trivial factor of N. Note that to argue the extracted square root of v differs from $\pm w$, subgroup $\langle g \rangle$ should contain at least another square root of v which is not $-w$. It cannot be $-w$ since -1 does not belong to $\langle g \rangle$ by construction. Such an element exists in $\langle g \rangle$ because, if the Jacobi symbol of g equals to -1 (or $(g|N) = -1$), $g^{p'q'}$ must be a non-trivial square root of unity in Z_N^*. As a consequence, $g^{p'q'} \cdot w$ is another square root of v that belongs to $\langle g \rangle$. Since the signing oracle and the break-in oracle never use w, the knowledge extractor allows to extract $g^{p'q'}$ with probability 1/2. The computations of the probability and the running time are identical to [6].

5 Experiment

We implement the proposed Fast-BM and Fast-AR schemes using openssl library in C. For comparison, we implement BM [3], AR [6], IR, optimized IR (IROpt) [4], and MMM [5]. We use GQ [8] as public key signature scheme in MMM. All schemes except MMM generate a short size signature of which size is k while MMM generates a signature of which size is $4k$, where k is the bit-length of RSA modulus N. The experiment is performed on Intel i5 2.6 GHz laptop with 16 GB RAM under OS X. The hash length (l) is fixed to 160 bit.

Figure 1 illustrates the key setup time, the signing time per message, the key update time, and the verification time per message by varying the number of periods T from 10 to 100000 denoted as x axis when the security key parameter k is 2048. The y axis represents the execution time in second.

Figure 1(a) shows the setup time. As T increases, the setup time becomes significantly large in IR and IROpt. For instance, it is 1,000 s when $T = 100{,}000$, and it will be 10,000 s when $T = 1{,}000{,}000$ in IR and IROpt while in Fast-BM and Fast-AR it is a few ten seconds in which a safe RSA is generated.

Figure 1(b) represents the signing time per message. The signing time is proportional to T in BM and AR while it is independent of T in IR, IROpt, MMM, Fast-BM and Fast-AR. Note that the signing time complexity is $O(k^3)$ in Fast-BM and Fast-AR where k represents the bit length of modulus N.

Figure 1(c) indicates the key update time. In BM, AR, MMM, Fast-BM, and Fast-AR, the key update time is constant only depending on k and l while it is proportional to T and $\log T$ in IR and IROpt, respectively.

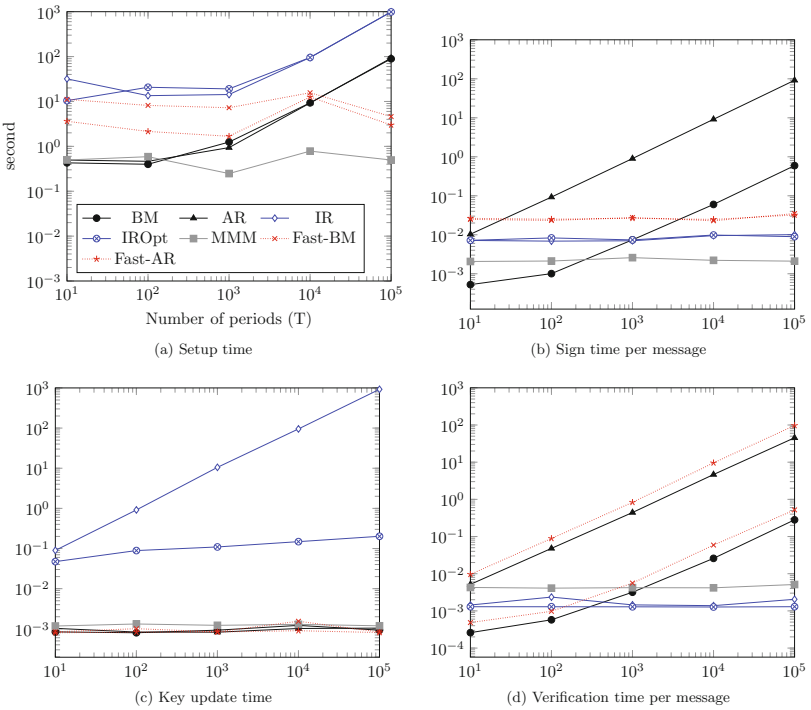
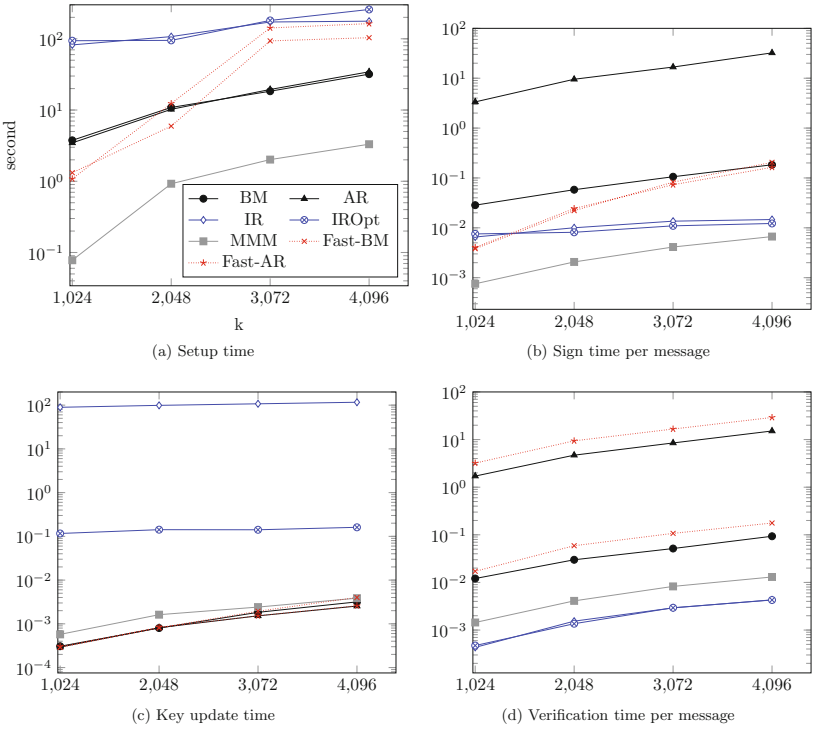


Fig. 1. Execution time variation as T varies when $l = 160$ and $k = 2048$

Figure 1(d) denotes the verification time. The verification time is proportional to T in BM, AR, Fast-BM and Fast-AR since T number of exponentiation operations are performed for verification in those algorithms, while it is irrelevant to T in IR, IROpt, and MMM.

Figure 2 illustrates the setup, signing, update, and verification times by varying k which is the bit-length of RSA modulus N when $l = 160$ and $T = 10000$. The setup time increases faster in IR, IROpt, Fast-BM, and Fast-AR since they use a safe RSA where $N = pq$, $p = 2p' + 1$, $q = 2q' + 1$, and p, q, p', q' are prime. When k is 2048, BM, AR, MMM, Fast-BM, and Fast-AR algorithms show a similar setup time.

Figure 2(b) shows the signing time per message. Since the signing time in all algorithms except Fast-BM and Fast-AR is proportional to k^2 if T and l are fixed, the signing time increases similarly in all algorithms as k increases. On the other hand, the time increases faster in Fast-BM and Fast-AR as k increases. Fast-AR is as fast as Fast-BM while AR is l time slower than BM since Fast-AR does not require $l(T + 1)$ squares.



(a) Setup time

(b) Sign time per message

(c) Key update time

(d) Verification time per message

Fig. 2. Execution time variation as k varies when $T = 10000$

Figure 2(c) shows the key update time. As k increases, the key update time increases in all schemes. In IR and IROpt, the key update time increases slowly since the effect of T against k decreases where the update time complexities in IR and IROpt are $O(k^2T)$ and $O(k^2 \log T)$ while they are $O(k^2)$ in the other algorithms.

Figure 2(d) represents the verification time. Regardless of k, IR and IROpt show the shortest verification time.

Table 1 summarizes the key sizes, setup time, signing time per message, key update time, and verification time in BM, AR, IR, IROpt, MMM, Fast-BM and Fast-AR. Fast-BM and Fast-AR reduce the signing time compared with BM and AR without sacrificing the other parameters.

Table 1. Comparison of BM, AR, IR, optimized IR, MMM, Fast-BM, and Fast-AR

Criteria	BM	AR	IR	IROpt	MMM	Fast-BM	Fast-AR
Public key size	$k(l+1)$	$2k$	$2k$		l	$k(l+1)$	$2k$
Secret key size	kl	k	$3k+l$	$k(\log(T)+2)+l$	$(\log l+\log t+2)l+2k$	kl	k
Signature size	$2k$	$k+l$	$k+2l$		$(\log l+\log t+2)l+4k$	$k+l$	$k+l$
Setup	$O(k^3+lTk^2)$	$O(k^3+lTk^2)$	$O(l^3T+k^3)$		$O(k^3+l^2\log l)$	$O(k^3+lk^2\log(T))$	$O(k^3+k^2\log(lT))$
Sign	$O(k^2(T+l))$	$O(k^2lT)$	$O(k^2l)$		$O(k^2l)$	$O(k^3)$	$O(k^3)$
Update	$O(k^2l))$	$O(k^2l)$	$O(k^2lT)$	$O(k^2l\log(T))$	$O(k^2l+l^2\log t)$	$O(k^2l)$	$O(k^2l)$
Verification	$O(k^2(T+l))$	$O(k^2lT)$	$O(k^2l)$		$O(k^2l)$	$O(k^2(T+l))$	$O(k^2lT)$
Security	Factoring		Strong RSA		One way hash	Factoring	

6 Related Work

The pioneering studies addressing the forward secure signatures were first proposed by Anderson [2] and subsequently formalized by Bellare and Miner in [3]. In a forward-secure signatures scheme, the forward- security property is attained by dividing time into T discrete intervals, and using a different secret key within each interval. The main challenge in designing forward-secure signature schemes is efficiency: an ideal scheme must have constant (public and secret) key sizes, constant signature size as well as constant signing, verification, and (public and secret) key update operations.

In the first category, the schemes use some generic method in which a master public key is used to certify the current public key for a particular time period (via a chain of certificates). Usually, these schemes increase storage space by noticeable factors in order to maintain the current (public) certificates and the (secret) keys for issuing future certificates. They also require longer verification times than ordinary signatures do, because the verifier needs to verify the entire certificate chain in addition to verifying the actual signature on the message. There is, in fact, a trade-off between storage space and verification time. These schemes include the tree-based scheme of Bellare and Miner [BM99] (requiring storage of about $O(\log T)$ secret keys and non-secret certificates, and verification of about $O(\log T)$ ordinary signatures), the scheme of Krawczyk [9] (requiring storage of T non-secret certificates, and verification of only 2 ordinary signatures), and the scheme of Malkin et al. [5] has constant-size public key while the secret key size, the signature size, signing and verifying time are $O(\log t)$ where t denotes the time interval index which is less than T. The scheme of Holt [10] has constant-size secret key and signatures but requires T non-secret certificates storage/communication to verify signatures. The generic construction proposed by Libert et al. [11] has a non-constant signature size and computational overhead and the exact complexities depend on the underlying schemes.

In the second category, the schemes are built upon standard signature schemes. The main advantage of these schemes is that they achieve better dependence on T. In particular, they typically have constant size parameters. The first such scheme is based on the Fiat-Shamir signature scheme [3]. Abdalla and Reyzin scheme [6] shortens secret and public keys of at the expense of signing and verifying time. Itkis and Reyzin scheme [4] has shorter signing and verifying time derived from the underlying Guillou-Quisquater signature scheme [8] at the expense of logarithmic key update time and the secret key size. Kozlov and Reyzin [12] propose another scheme based on a similar optimizing technique used in [4]. The scheme is an improved version of [4], and the key update time and the secret key size grow logarithmically in T. However, linear-time operations are needed at the beginning of each period in the scheme.

Boyen et al. [13] proposed a forward-secure signature scheme, where the secret key is encrypted with a second factor such as a user's password and can be updated in its encrypted form. The scheme in [13] based on [14] features a constant signing time at the expense of its key update time in $O(\log T)$, its secret key size in $O(\log^2 T)$, its public key size in $O(\log T)$, and comparatively stronger

cryptographic assumption. It makes use of a very specific mathematical setting consisting of groups equipped with a bilinear mapping whose computation is expensive. Abdalla et al. [15] proposed a variant of [4] to have a much tighter security reduction, however, assuming stronger security assumptions.

7 Conclusion

In this paper, we propose fast forward secure digital signature schemes called Fast-BM and Fast-AR which provide fast signing and key update with constant size public and secret keys, and a short constant size signature. The proposed schemes are applicable to real-time surveillance streaming applications as well as the traditional forward secure signature systems. In the proposed schemes, the signing and the key update are performed in $O(k^3)$ meaning that they are independent of the maximum period T, where k denotes the bit length of module N in RSA. In real implementation, the signing and the update can be performed within 25 ms in the proposed schemes regardless of T when $k = 2048$ while they require 200 ms in the optimized IR. The signature size is only 2240 bit when $k = 2048$, and $l = 160$ in our schemes. Fast-BM and Fast-AR schemes are secure under the factoring assumption in the random oracle model which is a weaker assumption than a strong RSA which IR is based on.

Acknowledgments. This research was supported by IT R&D program MKE/KEIT (No. 10041608, Embedded system Software for New-memory based Smart Device), by Next-Generation Information Computing Development Program through NRF funded by the Ministry of Science, ICT & Future Planning (No. NRF-2016M3C4A7937117), and by Basic Science Research Program through NRF funded by the Ministry of Education (No. 2016R1D1A1B03934545).

References

1. Bellare, M., Yee, B.: Forward-security in private-key cryptography. In: Joye, M. (ed.) CT-RSA 2003. LNCS, vol. 2612, pp. 1–18. Springer, Heidelberg (2003). doi:10.1007/3-540-36563-X_1
2. Anderson, R.: Two remarks on public-key cryptology - invited lecture. In: The Fourth ACM Conference on Computer and Communications Security (CCS) (1997)
3. Bellare, M., Miner, S.K.: A forward-secure digital signature scheme. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 431–448. Springer, Heidelberg (1999). doi:10.1007/3-540-48405-1_28
4. 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). doi:10.1007/3-540-44647-8_20
5. Malkin, T., Micciancio, D., Miner, S.: Efficient generic forward-secure signatures with an unbounded number of time periods. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 400–417. Springer, Heidelberg (2002). doi:10.1007/3-540-46035-7_27

6. 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). doi:10.1007/3-540-44448-3_10
7. Ong, H., Schnorr, C.: Fast signature generation with a fiat shamir-like scheme. In: Advances in Cryptology - EUROCRYPT 1990, Workshop on the Theory and Application of of Cryptographic Techniques, Aarhus, Denmark, 21–24 May 1990, Proceedings, pp. 432–440 (1990)
8. Guillou, L.C., Quisquater, J.-J.: A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory. In: Barstow, D., Brauer, W., Brinch Hansen, P., Gries, D., Luckham, D., Moler, C., Pnueli, A., Seegmüller, G., Stoer, J., Wirth, N., Günther, C.G. (eds.) EUROCRYPT 1988. LNCS, vol. 330, pp. 123–128. Springer, Heidelberg (1988). doi:10.1007/3-540-45961-8_11
9. Krawczyk, H.: Simple forward-secure signatures from any signature scheme. In: Proceedings of the 7th ACM Conference on Computer and Communications Security (CCS 2000), Athens, Greece, 1–4 November 2000, pp. 108–115 (2000)
10. Holt, J.E.: Logcrypt: forward security and public verification for secure audit logs. In: The proceedings of the Fourth Australasian Symposium on Grid Computing and e-Research (AusGrid 2006) and the Fourth Australasian Information Security Workshop (Network Security) (AISW 2006), Hobart, Tasmania, Australia, pp. 203–211, January 2006
11. Libert, B., Quisquater, J., Yung, M.: Forward-secure signatures in untrusted update environments: efficient and generic constructions. In: Proceedings of the 2007 ACM Conference on Computer and Communications Security (CCS 2007), Alexandria, Virginia, USA, 28–31 October 2007, pp. 266–275 (2007)
12. Kozlov, A., Reyzin, L.: Forward-secure signatures with fast key update. In: Cimato, S., Persiano, G., Galdi, C. (eds.) SCN 2002. LNCS, vol. 2576, pp. 241–256. Springer, Heidelberg (2003). doi:10.1007/3-540-36413-7_18
13. 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 (CCS 2006), Alexandria, VA, USA, 30 October–3 November 2006, pp. 191–200(2006)
14. Boneh, D., Boyen, X., Goh, E.-J.: Hierarchical identity based encryption with constant size ciphertext. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 440–456. Springer, Heidelberg (2005). doi:10.1007/11426639_26
15. Abdalla, M., Ben Hamouda, F., Pointcheval, D.: Tighter Reductions for Forward-Secure Signature Schemes. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 292–311. Springer, Heidelberg (2013). doi:10.1007/978-3-642-36362-7_19