Forward-Secure Signatures in Untrusted Update Environments: Efficient and Generic Constructions

Benoît Libert
UCL Crypto Group
Belgium

Jean-Jacques Quisquater
UCL Crypto Group
Belgium

Moti Yung
Columbia University
USA

ABSTRACT

Forward-secure signatures (FSS) prevent forgeries for past time periods when an attacker obtains full access to the signer's storage. To simplify the integration of these primitives into standard security architectures, Boyen, Shacham, Shen and Waters recently introduced the concept of forward-secure signatures with untrusted updates where private keys are additionally protected by a second factor (derived from a password). Key updates can be made on encrypted version of signing keys so that passwords only come into play for signing messages.

The scheme put forth by Boyen *et al.* relies on bilinear maps and does not require the random oracle. The latter work also suggested the integration of untrusted updates in the Bellare-Miner forward-secure signature and left open the problem of endowing other existing FSS systems with the same second factor protection.

This paper solves this problem by showing how to adapt the very efficient generic construction of Malkin, Micciancio and Miner (MMM) to untrusted update environments. More precisely, our modified construction - which does not use random oracles either - obtains a forward-secure signature with untrusted updates from any 2-party multi-signature in the plain public key model. In combination with Bellare and Neven's multi-signatures, our generic method yields implementations based on standard assumptions such as RSA, factoring or the hardness of computing discrete logarithms. Like the original MMM scheme, it does not require to set a bound on the number of time periods at key generation.

Categories and Subject Descriptors: E.3 [Data]: Data Encryption – Public Key Cryptosystems

General Terms: Design, Security, Performance.

1. INTRODUCTION

If not appropriately dealt with, key exposures are likely to ruin the efforts of the research community towards devising more sophisticated and secure cryptographic systems. They seem unavoidable in the modern age of ubiquitous computing with the ever-increasing use of mobile and unprotected devices. It is indeed generally much easier to break into users' private storage than to find out their secret by actual cryptanalytic techniques. Hence, it turns out that the only way to cope with such a threat is to contain the damage when private keys are exposed.

In the public key setting, the past recent years saw the exploration of various techniques addressing the problem by means of key evolving protocols where public keys remain fixed throughout the lifetime of schemes whereas private keys are updated at discrete time intervals. This line of research was initiated by Anderson's suggestion [3] of forward-secure mechanisms that aim at preserving the security of past time periods after a private key theft. Subsequently introduced key-insulated [15, 16] and intrusion-resilient [23] security paradigms strive to protect communications not only preceding, but also following break-ins by storing part of the key material in a separate device.

Anderson's original flavor of key evolving security was formalized by Bellare and Miner [4] who gave proper definitions of forward-secure signatures (FSS) and proposed two constructions. The first one was a generic method with logarithmic complexity in the number of periods built on any signature. Their second scheme extended Fiat-Shamir signatures [18] into a FSS scheme with signatures of constant (i.e. independent of the lifetime of the scheme) size but linear cost in signature generation and verification. This number theoretic method was improved by Abdalla-Reyzin [2] and Itkis-Reyzin [22], the latter work notably achieving optimal signing and verification at the expense of slower key updates using Guillou-Quisquater signatures [20]. Kozlov and Reyzin [26] finally showed another method with fast updates and a great online/offline efficiency. Meanwhile, forward security was also considered in special kinds of signature schemes [1, 35]. On the other hand non-trivial realizations of forward-secure public key encryption schemes remained elusive until the work by Canetti, Halevi and Katz [13] that was improved by Boneh, Boyen and Goh [10].

Among generic FSS schemes that start from any digital signature, Anderson's storage-demanding construction [3] was improved by Krawczyk [25] into a scheme requiring constant private storage (though the overall storage remained linear). Using Merkle trees [29] in a suitable fashion, Malkin, Micciancio and Miner [28] interestingly described another system with an essentially unbounded number of time periods: the maximal number of periods did not have to be set at key generation and the complexity of their scheme was rather (and moderately) depending on the number of past periods. Besides, their technique (often called MMM) was quite efficient. Not only did it outperform previous generic constructions, but it also beat number theoretic schemes in at least one metric when implemented with similar parameters. A practical evaluation of the efficiency of all these schemes can be found in [14].

Forward-secure signatures with untrusted updates. In many existing software environments (such as GNU-PG or S/MIME), private keys are additionally proctected by an extra secret which is possibly derived from a password. In order to facilitate the integration of forward-secure primitives into such existing software architectures, Boyen, Shacham, Shen and Waters [12] suggested a new a forward-secure signature where private keys are additionally shielded by a second factor. Their scheme allows for an automated update procedure of encrypted keys: the user holding the second factor does not have to intervene in operations where the update algorithm is programmed to move forward in time a blinded version of the key at the beginning of each period. The second factor is only needed for signing messages as in many typical implementations of digital signatures. Beyond the usual forward security requirement, such a scheme prevents an adversary just in possession of the encrypted key to forge signatures for past, current and future periods.

The compatibility of key-evolving signatures with a second factor protection surprisingly remained overlooked until [12]. When FSS schemes are designed for very fine time granularities, it is handy to leave the implemented software automatically carry out updates at pre-scheduled instants. In realistic settings however, key management techniques should take into account the possible weaknesses of the computing environment. In particular, key-evolving signatures should be endowed with a safeguard against maliciously controlled computing platforms. Otherwise, adversaries may be able to delay the clock of signers' computers and thereby obtain a key that should have been erased for instance.

While the usual model [4] of forward security captures one aspect of exposures (i.e. the user's storage), "untrusted updates" introduced in [12] deal with a potential exposure of the computing environment. In forward-secure signatures, a second factor protection thus especially strengthens signatures as evidence of the signer's intentionality of actually signing the message.

The concrete implementation of forward-secure signature with untrusted updates (FSS-UU) suggested in [12] enjoys a provable security in the standard model, as opposed to the random oracle model [5]. It simultaneously offers a very attractive efficiency, notably featuring constant-size signatures and at most log-squared complexity in other metrics. On the other hand, it makes use of a very specific mathematical setting consisting of groups equipped with a bilinear mapping (a.k.a. pairing) whose computation remains expensive (from twice to four times as slow as RSA in currently most optimized implementations). Boyen *et al.* [12] also showed how to simply obtain untrusted updates in the Bellare-Miner [4] factoring-based FSS scheme and the same key-blinding method is easily seen to apply to the Abdalla-Reyzin system [2] as well. Unfortunately, these methods both suffer from linear complexities for signing and key generation and directly applying the same idea to the Itkis-Reyzin scheme [22] removes its attractive performance advantages. The authors of [12] left open the problem of efficiently achieving untrusted updates in other existing forward-secure signatures.

Our contribution. We describe generic forward-secure signatures with untrusted updates. We first show that untrusted updates can be simply obtained from any traditional forward-secure signature. The idea is merely to sign a message twice: once using a classical FSS scheme and a second time using a regular (i.e. non forward-secure) digital signature, the private key of which is re-derived from a second factor at each signing operation instead of being stored. While very simple, this method induces a definite overhead and we will of course be after more efficient constructions.

Extending the above idea a little further, we construct FSS-UU schemes from bipartite multi-signatures [21]. Recall that these primitives are meant to allow several signers to jointly sign a common message. We start from any 2-party such signature satisfying appropriate security definitions and see it as a FSS-UU scheme with a single time period. We then show how to bootstrap it by applying the sum and product compositions of Malkin *et al.* [28] so as to obtain FSS-UU schemes with more periods.

More precisely, when applied to a secure bipartite multi-signature in Boldyreva's model [9], the iterated sum composition of [28] provides a FSS-UU system with logarithmic-size signatures. The basic idea of this construction is to apply the key updating technique of the sum composition to only one of the two parties in the multi-signature and keep the second key unchanged as a function of the second factor throughout all periods. We also suggest efficiency tradeoffs that can be obtained by combining this modified sum composition with non-generic FSS-UU schemes built on [4, 2].

In a second step, we show how to benefit from the full power of the MMM construction if we start from a secure 2-party signature in the plain public key model of Bellare and Neven [7]. By adapting the sum-product composition of [28], we interestingly obtain a generic construction of forward-secure signature with untrusted updates for a practically unbounded number of periods. When combined with recently suggested [7] multi-signatures built on Schnorr [34], Guillou-Quisquater [20], Fiat-Shamir [18] or Ong-Schnorr [31], our construction provides pairing-free schemes based on discrete logarithm, RSA and factoring that enjoy the same efficiency as traditional FSS signatures resulting from [28]. These concrete instantiations rely on the random oracle methodology [5] only because underlying signatures do: our extension of MMM does not introduce additional random oracle assumptions. Hence, standard model multi-signatures fitting the plain public key model of [7] would thus give rise to new FSS-UU schemes without random oracles. This yields answers to the open question, raised in [12], of how to efficiently provide existing forward-secure signatures with the untrusted update property.

In the forthcoming sections, we first recall definitions and security notions for FSS-UU schemes in section 2. The generic method for obtaining untrusted updates in any FSS scheme in detailed in section 3 and section 4 describes our efficient extension of the MMM method.

2. DEFINTIONS

A forward-secure signature scheme with untrusted updates (FSS-UU) is made of the following algorithms.

Keygen(λ, r, T)**:** on input of a security parameter λ, a random tape r and a number of time periods T, this randomized algorithm returns a public key PK, the initial encrypted signing key EncSK_0 and a random second factor secret decryption key DecK. The initial period number is set to 0.

CheckKey$(t, T, \mathsf{EncSK}_t, \mathsf{PK})$**:** is an algorithm used to check the well-formedness of the private key EncSK_t at period t. The output is $\top$ if the latter was correctly generated and $\perp$ otherwise.

Update$(t, T, \mathsf{EncSK}_t, \mathsf{PK})$**:** given a period number t and the corresponding encrypted key EncSK_t, this algorithm returns an encrypted key EncSK_{t+1} for the next period and erases EncSK_t. It does not need the second factor decryption key.

Sign$(t, T, \mathsf{EncSK}_t, \mathsf{DecK}, M, \mathsf{PK})$**:** takes as input a message M, a period number t, the matching encrypted key EncSK_t and the second factor decryption key DecK. It returns a signature σ. The period number t is part of the latter.

Verify$(t, T, \mathsf{PK}, M, \sigma)$**:** takes as input the public key PK, a period number t and a message M bearing an alleged signature σ. It outputs $\top$ if the signature is correct and $\perp$ otherwise.

In these syntactic definitions, the validity test $\mathsf{CheckKey}$ aims at completely validating a newly generated encrypted key before erasing the old key. In practice, a check of EncSK by the signing algorithm suffices and an additional validity test at each update should only be performed at the signer's discretion to make sure that the signing process will not be disrupted in the new period.

The obvious completeness requirement imposes that properly generated signatures are always accepted by the verification algorithm.

The security model of [12] considers two orthogonal definitions that are both inspired from the well-known concept of chosen-message security [19]. The first one considers a game extending the usual notion of forward security as defined by Bellare-Miner [4]. In the game that the adversary plays against her challenger, she should be unable to forge a signature pertaining to an unexposed stage even knowing the second factor decryption key DecK.

Definition 1. *The* ***forward security*** *notion captures the negligible advantage of any PPT adversary in this game.*

1. *The challenger runs the key generation algorithm and gives the public key* PK *and the second factor decryption key* DecK *to the forger* $\mathcal{F}$*. The initial period number* t *is set to* 0.

2. $\mathcal{F}$ *adaptively interacts with the following oracles.*

   · **Sign** : *at any time, the forger can ask for a signature on an arbitrary message* M *for the current time period* t.

   · **Update** : *once she decides to move forward in time, the adversary queries the challenger that runs the update algorithm and increments the period number* t.

   · **Break-in** : *at some period, the forger enters the break-in phase and requests the challenger to reveal the current encrypted signing key* EncSK_t.

3. $\mathcal{F}$ *comes up with a message* $M^\star$ *and a signature* $\sigma^\star$ *for some period* $t^\star$*. If* t' *denotes the time period where the break-in query was made,* $\mathcal{F}$ *is declared successful provided* $\mathsf{Verify}(t^\star, T, \mathsf{PK}, M^\star, \sigma^\star) = 1$, $t^\star < t'$ *and* $M^\star$ *was not queried for signature at period* $t^\star$.

$\mathcal{F}$*'s advantage* $\mathsf{AdvFS}(\mathcal{F})$ *is her probability of victory taken over all coin tosses. We say that she* $(t, q_s, q_u, \varepsilon)$*-breaks the scheme if she has advantage* ε *within running in time* t *after* q_s *signing queries and* q_u *update queries.*

The second security notion, termed *update security*, captures the security against an adversary obtaining encrypted signing keys for all periods but not the second factor decryption key. It mirrors the fact that, at any time, the encrypted key EncSK is by itself useless to generate signatures.

Definition 2. *The* ***update security*** *property is the negligible advantage of a PPT adversary in this game.*

1. *The challenger performs the key generation and gives the public key* PK *and the initial encrypted key* EncSK_t *for period* $t = 0$ *to* $\mathcal{F}$*. The second factor decryption key* DecK *is held back from* $\mathcal{F}$.

2. $\mathcal{F}$ *adaptively interacts with the following oracles.*

   · **Sign** : *at any time, the forger can ask for a signature on an arbitrary message* M *for the current time period* t.

   · **Update** : *once she decides to move forward in time, the adversary queries the challenger that runs the update algorithm and increments the period number* t.

3. $\mathcal{F}$ *outputs a message* $M^\star$ *and a signature* $\sigma^\star$ *for some period* $t^\star$*. She wins if* $M^\star$ *was not queried for signature at period* $t^\star$ *and* $\mathsf{Verify}(t^\star, T, \mathsf{PK}, M^\star, \sigma^\star) = 1$.

$\mathcal{F}$*'s advantage* $\mathsf{AdvUS}(\mathcal{F})$ *is defined as in definition 1.*

3. ACHIEVING UNTRUSTED UPDATES IN ANY FORWARD-SECURE SIGNATURE

In the implementation proposed in [12], the second factor DecK is not taken as input by the key generation algorithm but is uniformly chosen by the latter in a set which is as large as the private key space. It is assumed to be in turn encrypted using a password that has sufficient entropy to prevent offline dictionary attacks. Hence, this second factor can also be the random seed used to generate a key pair for an ordinary signature scheme. It follows that a forward-secure signature $\Pi^{\mathsf{FS}} = (\mathsf{Keygen}^{\mathsf{FS}}, \mathsf{Update}^{\mathsf{FS}}, \mathsf{Sign}^{\mathsf{FS}}, \mathsf{Verify}^{\mathsf{FS}})$ can always be endowed with a second factor protection by combining them with a regular (i.e. non-forward-secure) digital signature $\Theta = (\mathcal{K}, \mathcal{S}, \mathcal{V})$. The public key PK of the FSS-UU scheme thus includes the public key $\mathsf{PK}^{\mathsf{FS}}$ of Π^{FS} as well as the public key pk of Θ. At any period, the "encrypted" signing key EncSK_t is the private key SK_t of Π^{FS} while the second factor DecSK is the seed used to generate (sk, pk). A signature on a message M is then given by the concatenation $\sigma = \langle \mathsf{Sign}^{\mathsf{FS}}(M, t, \mathsf{SK}_t), \mathcal{S}_{sk}(t||M) \rangle$ of both signatures. Verification is obviously achieved by running both verification algorithms.

The security of the FSS-UU scheme in the sense of definition 1 directly follows from the forward security of Π^{FS} while the update security, according to definition 2, is easily seen to rely on the unforgeability of Θ against chosen-message attacks [19].

We note that the construction is similar to another one used in [16] in a related context to obtain strongly key-insulated signatures. Combined with known results [4, 28], it shows - not so surprisingly - the existence of forward-secure signatures with untrusted updates and at most logarithmic complexity if digital signatures exist at all, which amounts to assuming that one-way functions exist [33].

Although clearly not optimal from an efficiency point of view, this method provides a fairly practical realization of untrusted updates in the Itkis-Reyzin forward-secure signature [22] at the expense of almost doubling the signature length. Recall that the latter scheme uses the Guillou-Quisquater [20] signature with a different prime exponent (that is part of the signature) for each time period. Untrusted updates can be achieved by including an independent prime GQ exponent e as well as a power $I = \mathsf{DecK}^e \bmod N$ in the public key (the same RSA modulus N being used in both schemes after the erasure of its prime factors). A signature then includes an Itkis-Reyzin signature as well as a proof of knowledge of DecK, a single Fiat-Shamir-like [18] hash value acting as a challenge in both non-interactive proofs in order to decrease the signature length.

4. EFFICIENT GENERIC CONSTRUCTIONS FROM 2-PARTY SIGNATURES

In this section, we construct a FSS-UU scheme from two-party multi-signatures [21] by extending the generic forward-secure signatures of Malkin, Micciancio and Miner [28].

Formally, a 2-party multi-signature scheme consists of a 4-uple $\mathsf{2MS} = (\mathsf{PG}^{\mathsf{MS}}, \mathsf{Kg}^{\mathsf{MS}}, \mathsf{Sign}^{\mathsf{MS}}, \mathsf{Verify}^{\mathsf{MS}})$ of algorithms. Among these, $\mathsf{PG}^{\mathsf{MS}}$ is a common parameter generation algorithm run by a central authority; $\mathsf{Kg}^{\mathsf{MS}}$ is a user key generation algorithm independently run by each signer; $\mathsf{Sign}^{\mathsf{MS}}$ is a possibly interactive algorithm jointly run by both parties (each of which taking his private key and the peer party's public key in addition to the message and public parameters) to sign a message while $\mathsf{Verify}^{\mathsf{MS}}$ is the verification algorithm allowing for the verification of a signature w.r.t. both parties' public keys. We emphasize that 2-party multi-signatures are a more general primitive than 2-party sequential multi-signatures defined in [17] in that the signing process is not necessarily non-interactive or sequential.

For the applications that we have in mind, we do not need a multi-signature scheme with a complex key generation protocol such as the one of [30]. Actually, the first generic composition (termed 'sum composition' in [28]) does not even require to consider a strong model as in [7]. With this first composition, we can settle for schemes that are secure in the *registered public key model* as defined in [9, 27]. In the latter, the adversary is challenged on a single public key belonging to a honest signer and attempts to frame the latter and wrongly accuse him of having jointly signed a message with corrupt users. The attacker is allowed to generate herself public keys for corrupt signers. The registered public key model deals with rogue key attacks by requiring the adversary to reveal the matching private keys when registering public keys that she creates for herself.

The second composition, called 'product composition' in [28], is more difficult to adapt in the present context and its extension does not appear to work in a fully generic and black box fashion. Nevertheless, we show that it can be applied if one of its component is a FSS-UU scheme obtained by means of the sum composition applied to a bipartite multi-signature in the *plain public key model* of Bellare and Neven [7].

4.1 The Sum Composition

The sum composition of [28] is extended so as to provide a FSS-UU scheme with $2T$ periods from two such schemes having each T periods. When iterated $\log T$ times, it turns a 2-party multi-signature (seen as FSS-UU with a single period) into a forward-secure signature with untrusted updates over $T = 2^\tau$ periods that will be dubbed FSS-UU$^\oplus$ here.

The idea of the generic construction is to use two FSS-UU schemes $\Sigma_0 = (\mathsf{Keygen}_0, \mathsf{CheckKey}_0, \mathsf{Update}_0, \mathsf{Sign}_0, \mathsf{Verify}_0)$ and $\Sigma_1 = (\mathsf{Keygen}_1, \mathsf{CheckKey}_1, \mathsf{Update}_1, \mathsf{Sign}_1, \mathsf{Verify}_1)$ that are both implemented with the same second factor decryption key DecK used as a global variable by algorithms Sign_0 and Sign_1. When the construction is applied $\log T$ times to obtain T periods, the second factor DecK is chosen once-and-for-all at the highest level setup and used as second factor in both Σ_0 and Σ_1. When combined FSS-UU schemes Σ_0 and Σ_1 have only one period (that is, 2-party multi-signatures $\mathsf{2MS}_0$ and $\mathsf{2MS}_0$), DecK is used as a random seed to generate the key pair $(\tilde{sk}, \tilde{pk}) \leftarrow \mathsf{Kg}^{\mathsf{MS}}(\mathsf{DecK})$ to be used as the second party key in both $\mathsf{2MS}_0$ and $\mathsf{2MS}_1$.

4.1.1 *Description*

Our notations are close to the ones of [28] and we use the same ingredients. Namely, G denotes a length-doubling forward-secure pseudorandom generator[1] (as originally suggested in [25]) and H is a collision-resistant hash function. Finally, ε and $a||b$ respectively stand for the empty string and the concatenation of two binary strings a and b. When subscripted by binary indexes or not, r always denotes a random seed. We assume that seeds r, r_0, r_1 and second factors DecK are random strings of equal length.

At a high level, this construction starts from a bipartite multi-signature $\mathsf{2MS}$ and recursively applies the sum composition technique of [28] to generate keys for only one of the two parties. The second party's key pair is remains unchanged throughout all periods and is derived from a second factor DecK chosen in the setup phase. This factor is kept as a global variable in all recursive steps and only comes into play for signing messages.

At the setup of the scheme, the key generation algorithm is called with arguments including $T = 2^\tau$ and $pk' = \varepsilon$. As in [28], $\mathsf{SKeygen}$ and $\mathsf{PKeygen}$ denote algorithms carrying out the same operations as Keygen but respectively erase public and private outputs.

[1]A forward-secure PRG is one where seeds are periodically refreshed and where the exposure of the seed at a given period leaks no computable information about pseudorandom sequences generated in past periods. Efficient constructions of such generators from regular PRGs exist [8].

```
$\mathsf{Keygen}(\lambda, r, T, pk')$
    If $(pk' = \varepsilon$ and $T > 1)$            /* initialization */
      $\mathsf{cp} \leftarrow \mathsf{PG}^{\mathsf{MS}}(\lambda)$;
      $(r, \mathsf{DecK}) \leftarrow G(r)$;
          /* $\mathsf{DecK}$ is kept as a static global variable */
      $(\tilde{sk}, \tilde{pk}) \leftarrow \mathsf{Kg}^{\mathsf{MS}}(\mathsf{DecK})$;
      $pk' \leftarrow (\mathsf{cp}, \tilde{pk})$;
    endif
    If $(T = 1)$
      $(sk, pk) \leftarrow \mathsf{Kg}^{\mathsf{MS}}(r)$;
      Return
        $\big(\mathsf{EncSK} = \langle sk, 0, pk, \varepsilon \rangle, \mathsf{PK} = (pk', H(pk, \varepsilon))\big)$;
    else
      $(r_0, r_1) \leftarrow G(r)$;
      $(\mathsf{EncSK}_0, \mathsf{PK}_0) \leftarrow \mathsf{Keygen}_0(\lambda, r_0, T/2, pk')$;
      $\mathsf{PK}_1 \leftarrow \mathsf{PKeygen}_1(\lambda, r_1, T/2, pk')$;
      Parse $\mathsf{PK}_0$ as $(pk', \mathsf{pk}_0)$ and $\mathsf{PK}_1$ as $(pk', \mathsf{pk}_1)$;
      $\mathsf{PK} \leftarrow (pk', H(\mathsf{pk}_0, \mathsf{pk}_1))$;
    endif
    Return $(\underbrace{\langle \mathsf{EncSK}_0, r_1, \mathsf{pk}_0, \mathsf{pk}_1 \rangle}_{\mathsf{EncSK}}, \mathsf{PK})$;
```

```
$\mathsf{CheckKey}\big(t, T, \overbrace{\langle \mathsf{EncSK}', r_1, \mathsf{pk}_0, \mathsf{pk}_1 \rangle}^{\mathsf{EncSK}}, \mathsf{PK}\big)$
   Parse $\mathsf{PK}$ as $(pk', h)$;
   Output $\perp$ if $h \neq H(\mathsf{pk}_0, \mathsf{pk}_1)$;
   Set $\mathsf{PK}_0 = (pk', \mathsf{pk}_0)$ and $\mathsf{PK}_1 = (pk', \mathsf{pk}_1)$;
   If $(T = 1)$
      Let $pk = \mathsf{pk}_0$ and $sk = \mathsf{EncSK}'$;
      Return $\perp$ if $r_1 \neq 0$, $\mathsf{pk}_1 \neq \varepsilon$ or $(sk, pk)$
      is not a valid key pair for 2MS;
      Return $\top$;
   endif
   If $(t < T/2)$
      return $\mathsf{CheckKey}_0\big(t, T/2, \mathsf{EncSK}', \mathsf{PK}_0\big)$;
   else
       return $\mathsf{CheckKey}_1\big(t - T/2, T/2, \mathsf{EncSK}', \mathsf{PK}_1\big)$;
   endif
```

```
$\mathsf{Update}\big(t, T, \overbrace{\langle \mathsf{EncSK}', r_1, \mathsf{pk}_0, \mathsf{pk}_1 \rangle}^{\mathsf{EncSK}}, \mathsf{PK}\big)$
   If $(t = T - 1)$ erase EncSK and return "no period left";
   Parse $\mathsf{PK}$ as $(pk', h)$;
   Output $\perp$ if $h \neq H(\mathsf{pk}_0, \mathsf{pk}_1)$;
   Set $\mathsf{PK}_0 = (pk', \mathsf{pk}_0)$ and $\mathsf{PK}_1 = (pk', \mathsf{pk}_1)$;
   If $(t + 1 < T/2)$
       $\mathsf{EncSK}' \leftarrow \mathsf{Update}_0\big(t, T/2, \mathsf{EncSK}', \mathsf{PK}_0\big)$;
   else
      If $(t + 1 = T/2)$
          $\mathsf{EncSK}' \leftarrow \mathsf{SKeygen}_1(\lambda, r_1, T/2, pk')$; $r_1 \leftarrow 0$;
      else
          $\mathsf{EncSK}' \leftarrow \mathsf{Update}_1\big(t - T/2, T/2, \mathsf{EncSK}', \mathsf{PK}_1\big)$;
      endif
   endif
```

```
$\mathsf{Sign}\big(t, T, \overbrace{\langle \mathsf{EncSK}', r_1, \mathsf{pk}_0, \mathsf{pk}_1 \rangle}^{\mathsf{EncSK}}, \mathsf{DecK}, M, \mathsf{PK}\big)$
   At the initial stage of the recursion, set $M \leftarrow M||t$;
   Parse $\mathsf{PK}$ as $(pk', h)$ and $pk'$ as $(\mathsf{cp}, \tilde{pk})$;
   Output $\perp$ if $h \neq H(\mathsf{pk}_0, \mathsf{pk}_1)$;
   Set $\mathsf{PK}_0 = (pk', \mathsf{pk}_0)$ and $\mathsf{PK}_1 = (pk', \mathsf{pk}_1)$;
   If $(T = 1)$
      Let $pk \leftarrow \mathsf{pk}_0$ and $sk \leftarrow \mathsf{EncSK}'$;
      Return $\perp$ if $r_1 \neq 0$, $\mathsf{pk}_1 \neq \varepsilon$ or $(sk, pk)$
      is not a valid key pair for 2MS;
      $(\tilde{sk}, \tilde{pk}) \leftarrow \mathsf{KG}^{\mathsf{MS}}(\mathsf{DecK})$;
      Return $\perp$ if $\tilde{pk}$ differs from the 2nd part of $pk'$;
      Run $\mathsf{Sign}^{\mathsf{MS}}$ on $M$ on behalf of both parties
      using $sk$, $\tilde{sk}$ and $\mathsf{cp}$ to generate $\sigma'$;
      Return $\langle \sigma', pk, \varepsilon \rangle$;
   endif
   If $(t < T/2)$
      $\sigma' \leftarrow \mathsf{Sign}_0(t, T/2, \mathsf{EncSK}', \mathsf{DecK}, M, \mathsf{PK}_0)$
   else
      $\sigma' \leftarrow \mathsf{Sign}_1(t - T/2, T/2, \mathsf{EncSK}', \mathsf{DecK}, M, \mathsf{PK}_1)$
   Return $(\underbrace{\langle \sigma', \mathsf{pk}_0, \mathsf{pk}_1 \rangle}_{\sigma}, t)$
```

```
$\mathsf{Verify}\big(t, T, \mathsf{PK}, M, \overbrace{\langle \sigma', \mathsf{pk}_0, \mathsf{pk}_1 \rangle}^{\sigma}\big)$
   Parse $\mathsf{PK}$ as $(pk', h)$ and $pk'$ as $(\mathsf{cp}, \tilde{pk})$;
   Output $\perp$ if $h \neq H(\mathsf{pk}_0, \mathsf{pk}_1)$;
   If $(T = 1)$
       Return $\perp$ if $\mathsf{pk}_1 \neq \varepsilon$;
       Return $\mathsf{Verify}^{\mathsf{MS}}(\mathsf{cp}, \mathsf{pk}_0, \tilde{pk}, M, \sigma')$;
   endif
   Set $\mathsf{PK}_0 = (pk', \mathsf{pk}_0)$ and $\mathsf{PK}_1 = (pk', \mathsf{pk}_1)$;
   If $(t < T/2)$
      Return $\mathsf{Verify}_0(t, T/2, \mathsf{PK}_0, M, \sigma')$;
   else
      Return $\mathsf{Verify}_1(t - T/2, T/2, \mathsf{PK}_1, M, \sigma')$;
```

4.1.2 Security

To prove the update security of FSS-UU$^{\oplus}$, we could simply show that a 2-party signature can be seen as a FSS-UU with one period and separately prove that the sum composition of any two FSS-UU schemes (such as the one described in [12] for instance) using the same second factor yields another such scheme with more periods. However, in the proof of update security (theorem 4.4) for the second composition, we will use the details of the proof of the next theorem. Moreover, we obtain a more efficient (i.e. independent of T in terms of probability) reduction by analyzing the security of the whole iterated composition as we do here.

THEOREM 4.1. *If the underlying* 2*-party multi-signature* 2MS *is secure against chosen-message attacks in the registered public key model, the sum composition provides update security. Namely, an update security adversary $\mathcal{F}$ implies a chosen-message attacker $\mathcal{B}$ having identical advantage* $\mathsf{AdvUS}(\mathcal{F})$ *over* 2MS *within comparable running time.*

PROOF. We show an algorithm $\mathcal{B}$ mounting a chosen-message attack against of a 2-party multi-signature scheme $2\mathsf{MS} = (\mathsf{PG}^{\mathsf{MS}}, \mathsf{Kg}^{\mathsf{MS}}, \mathsf{Sign}^{\mathsf{MS}}, \mathsf{Verify}^{\mathsf{MS}})$ in the registered public key model by interacting with an adversary $\mathcal{F}$ against the update security of FSS-UU$^{\oplus}$. Algorithm $\mathcal{B}$ faces a challenger $\mathcal{C}$ that hands her a public key $\tilde{pk}$ for 2MS.

At any time, $\mathcal{B}$ is allowed to register arbitrary public keys

pk and is then requested to reveal the matching private key sk. She may also trigger a joint execution of the signing protocol with $\mathcal{C}$, the latter running the 2-party protocol on behalf of the honest signer $\tilde{pk}$ while she plays the role of the (registered) corrupt signer pk.

$\mathcal{F}$'s input consists of a public key for the iterated sum composition over T periods. The latter is generated following almost exactly the specification of the scheme. Namely, $\mathcal{B}$ recursively runs Keygen but defines $pk' = (\mathsf{cp}, \tilde{pk})$ using her own challenge public key $\tilde{pk}$ instead of generating a key pair $(\tilde{sk}, \tilde{pk})$ from a second factor DecK. In other words, at the first run of Keygen, $\mathcal{B}$ skips line 5 of algorithm Keygen and defines pk' using her own input at line 6. Since DecK only comes into play at the initial execution of Keygen, recursive calls of the latter go through as in the real game. Whenever $\mathcal{B}$ needs to create a new key pair (sk, pk) for the $\mathsf{2MS}$ scheme, she registers it in the game that she plays against her own challenger $\mathcal{C}$ and discloses sk. The whole key generation entails the registration of $\log T$ key pairs for $\mathsf{2MS}$. Eventually, the initial "encrypted" private key EncSK_0 for period 0 in FSS-UU$^{\oplus}$ is obtained and given to $\mathcal{F}$ who starts issuing queries.

Update queries: whenever $\mathcal{F}$ wants to move to the next time period, $\mathcal{B}$ simply runs the update algorithm that does not need the second factor DecK (which is unknown to $\mathcal{B}$). All "encrypted" private key elements that ever have to be actually stored by a signer (that is, all keys but the private key $\tilde{sk}$ that matches $\tilde{pk}$ and is presumably derived from the second factor) are computable by $\mathcal{B}$ that can perfectly answer update queries.

Signing queries: at any period t, $\mathcal{F}$ may query her challenger $\mathcal{B}$ to sign a message M. To answer such a request, $\mathcal{B}$ triggers the recursive signing algorithm and follows its specification before the last step of the recursion (when $T = 1$). Upon entering the latter stage, the "encrypted" private key has the shape $\mathsf{EncSK} = \langle sk, 0, pk, \varepsilon \rangle$ where (sk, pk) is a valid key pair for $\mathsf{2MS}$ and must have been registered to $\mathcal{C}$ by construction. Since (sk, pk) was registered, $\mathcal{B}$ may query her challenger $\mathcal{C}$ to obtain a multi-signature σ' on M w.r.t public keys pk and $\tilde{pk}$. The triple $(\sigma', pk, \varepsilon)$ is set as the result of the final call in the recursion and allows completing the signature generation.

Forgery: eventually, a successful adversary $\mathcal{F}$ is expected to forge a valid signature $\sigma^\star = (\langle \sigma'^\star, \mathsf{pk}_0^\star, \mathsf{pk}_1^\star \rangle, t^\star)$ on a message $M^\star$ for a period $t^\star$ during which $M^\star$ was not the input of a signing query. Then, $\mathcal{B}$ executes the recursive verification and key checking algorithms until reaching the final stage $T = 1$ where the "encrypted" key has the shape $\mathsf{EncSK} = \langle sk, 0, pk, \varepsilon \rangle$. Note that, unless a collision is found on the hash chain ending with $H(\mathsf{pk}_0^\star, \mathsf{pk}_1^\star)$ (which is part of the public key of FSS-UU$^{\oplus}$), the pair (sk, pk) must have been registered by construction. Since $\mathcal{F}$ did not query $M^\star$ for signature at period $t^\star$, $\mathcal{B}$ did not query $\mathcal{C}$ for obtaining a 2-party multi-signature on the augmented message $M^\star || t^\star$ (recall that the signing algorithm appends the period number to the message at the initial recursion stage). It follows that $\mathcal{B}$ wins against $\mathcal{C}$ by outputting $M^\star || t^\star$ along with the pair (sk, pk) and the 2-party signature embedded in the inner part of $\sigma'^\star$. $\square$

Theorem 4.2. *If the* $\mathsf{2MS}$ *scheme is secure against chosen-message attacks, the FSS-UU*$^{\oplus}$ *construction provides forward security. Namely, for an instantiation of FSS-UU*$^{\oplus}$ *over* T *periods, an adversary* $\mathcal{F}$ *reaching advantage* $\mathsf{AdvFS}(\mathcal{F})$ *within running time* t *after* q_s *and* q_u *signing and update queries implies a chosen-message attacker with advantage* $\mathsf{AdvFS}(\mathcal{F})/T$ *over* $\mathsf{2MS}$ *within time* $t' \leq t + q_s t_{\mathsf{Sng}}^{\mathsf{MS}} + q_u t_{\mathsf{Kg}}^{\mathsf{MS}}$*, where* $t_{\mathsf{Sng}}^{\mathsf{MS}}$ *and* $t_{\mathsf{Kg}}^{\mathsf{MS}}$ *respectively denote the time complexities of signing and key generation algorithms in* $\mathsf{2MS}$.

Proof. The result almost directly follows from the result of Malkin *et al.* for the sum composition (i.e. theorem 3 in [28]) since a FSS-UU scheme is nothing but a traditional forward-secure signature when the adversary gets to know the second factor DecK as in the game of definition 1. However, the construction combines 2-party multi-signature schemes (instead of regular digital signatures) at the bottom of the recursion. To be complete, we must prove that 2-party multi-signatures indeed implement single-period FSS-UU schemes at the last step of the signing algorithm.

An adversary $\mathcal{F}$ against the forward security of a FSS-UU with one period is given the public key PK and the second factor DecK but is not allowed to make a break-in query. We thus consider a forger $\mathcal{B}$ against a $\mathsf{2MS}$ scheme that receives a public key pk^{MS} from her challenger $\mathcal{C}$. Her goal is to use $\mathcal{F}$ to break the security of $\mathsf{2MS}$. At the beginning of the game that she plays against $\mathcal{C}$, she registers the key pair $(\tilde{sk}, \tilde{pk}) \leftarrow \mathsf{Kg}^{\mathsf{MS}}(\mathsf{DecK})$ for a randomly chosen second factor DecK. She then defines $pk' = (\mathsf{cp}, \tilde{pk})$ (where cp are public parameters of $\mathsf{2MS}$ obtained from $\mathcal{C}$) and starts $\mathcal{F}$ on input of a public key $\mathsf{PK} = (pk', H(pk^{\mathsf{MS}}, \varepsilon))$ and DecK. The "encrypted private key" is implicitly defined as $\mathsf{EncSK} = \langle sk^{\mathsf{MS}}, 0, pk^{\mathsf{MS}}, \varepsilon \rangle$ (but $\mathcal{B}$ does not know sk^{MS}).

Whenever $\mathcal{F}$ asks for a signature on a message M, $\mathcal{B}$ queries her own challenger $\mathcal{C}$ to obtain a 2-party signature σ' on M for signers pk^{MS} and $\tilde{pk}$. Note that she can run the joint signing protocol on behalf of the latter since she knows $\tilde{sk}$. Having received σ', she hands the signature $\langle \sigma', pk^{\mathsf{MS}}, \varepsilon \rangle$ back to $\mathcal{F}$.

After polynomially many queries, $\mathcal{F}$ outputs a forgery $\langle \sigma'^\star, pk^{\star\mathsf{MS}}, \varepsilon \rangle$ for a new message $M^\star$. Unless H is not collision-resistant, we must have $pk^{\star\mathsf{MS}} = pk^{\mathsf{MS}}$ and $\sigma'^\star$ must be a valid multi-signature on $M^\star$ for public keys pk^{MS} and $\tilde{pk}$. It obviously follows that $\mathcal{B}$ succeeds whenever $\mathcal{F}$ does.

The result and claimed bounds directly derive from theorem 3 in [28], which implies that the sum of two FSS schemes with T periods yiels a FSS schemes with $2T$ stages. $\square$

We note that theorems 4.1 and 4.2 remain true if FSS-UU$^{\oplus}$ is applied to a multi-signature scheme in the model of [7] where forgers are not required to register public keys that they create for themselves or to prove the knowledge of their private key. Any secure multi-signature in the model of [7] is indeed also secure in the registered public key model.

However, considering the latter allows us to securely instantiate FSS-UU$^{\oplus}$ with Boldyreva's short multi-signature [9], that extends Boneh-Lynn-Shacahm signatures [11], or the standard model scheme of Lu *et al.* [27], which builds on the Waters signature [36].

In the latter case, we obtain an interesting alternative to [12] in the standard model. It indeed enjoys a security resting on the classical Diffie-Hellman assumption in bilinear map groups instead of the related assumption used in [12], the strength of which logarithmically depends on the number of periods (that only affects the reduction cost here). Verification is also faster than in [12] since it entails two bilinear map evaluations (instead of 3). The disadvantages are the length of signatures, that contain $\log T$ hash values[2], and the slower key generation (which is linear in T).

4.1.3 Efficiency Tradeoffs with Hybrid Schemes

The above sum composition admits several efficiency tradeoffs if we combine it with appropriate concrete FSS-UU schemes with more than one time period at the final stage of the iterated composition.

For instance, we can obtain signatures of sub-logarithmic size at the expense of a logarithmic complexity for signing and verifying. This is possible using the following extension of the Abdalla-Reyzin number theoretic scheme [2]. The recursive key generation algorithm initially generates a Blum integer $N = pq$ (i.e. a product of large primes p, q such that $p = q = 3 \bmod 4$) and discards its factorization. It also chooses a second factor $\mathsf{DecK} \xleftarrow{R} \mathbb{Z}_N^*$ and sets $\tilde{pk} = \mathsf{DecK}^{2^{\ell(s+1)}} \bmod N$ and $pk' = (N, \tilde{pk})$ where $s = \log T$ and ℓ is a security parameter. We have to replace multi-signatures at recursive final steps with an instance of the following FSS-UU scheme inspired from the one suggested in section 6 of [12].

$\mathsf{Keygen}(\ell, r, s)$: randomly choose $S_0 \xleftarrow{R} \mathbb{Z}_N^*$ and compute $U = S_0^{2^{\ell(s+1)}} \bmod N$. Set $\mathsf{EncSK} = \langle S_0, 0, U, \varepsilon\rangle$ and $\mathsf{PK} = (pk', H(U, \varepsilon))$.

$\mathsf{CheckKey}(t, s, \mathsf{EncSK}, \mathsf{PK})$: parse EncSK as $\langle S_t, 0, U, \varepsilon\rangle$. Return $\top$ if $U = S_t^{2^{\ell(s+1-t)}} \bmod N$ and $\perp$ otherwise.

$\mathsf{Update}(t, s, \mathsf{EncSK}, \mathsf{PK})$: parse EncSK as $\langle S_t, 0, U, \varepsilon\rangle$. Set $S_{t+1} = S_t^{2^{\ell}} \bmod N$ and return $\langle S_{t+1}, 0, U, \varepsilon\rangle$.

$\mathsf{Sign}(t, s, \mathsf{EncSK}, \mathsf{DecK}, M, \mathsf{PK})$: parse EncSK as $\langle S_t, 0, U, \varepsilon\rangle$ and conduct the following steps.

1. Pick $R \xleftarrow{R} \mathbb{Z}_N^*$ and set $Y \leftarrow R^{2^{\ell(s+1-t)}} \bmod N$.
2. Compute $\omega = H'(Y, M) \in \{0,1\}^{\ell}$ using a random oracle H'.
3. Unblind the key as $S'_t \leftarrow S_t \cdot \mathsf{DecK}^{2^{\ell t}} \bmod N$ and compute $Z \leftarrow R \cdot S_t'^{\omega} \bmod N$.

Return $\langle \sigma' = (\omega, Z), U, \varepsilon\rangle$.

$\mathsf{Verify}(t, s, \mathsf{PK}, M, \sigma)$: parse σ as $\langle \sigma' = (\omega, Z), U, \varepsilon\rangle$ and PK as $(pk', H(U, \varepsilon))$ where $pk' = (N, \tilde{pk})$.

1. Compute $Y' \leftarrow Z^{2^{\ell(s+1-t)}} \cdot (\tilde{pk} \cdot U)^{-\omega} \bmod N$.
2. Return $\top$ if $\omega = H'(Y', M)$ and $\perp$ otherwise.

Note that hashing the local index t along with the pair (Y, M) to generate the Fiat-Shamir-like challenge using H' is useless since the global period number was already appended to M at the first recursive call to signing algorithm.

With $s = \log T$, the above concrete scheme can be plugged into the sum composition applied $\log T - \log\log T$ times so as to obtain a hybrid concrete/generic scheme over T periods with logarithmic complexity for signing and verifying while signatures contain $O(\log T - \log\log T)$ hash values. Signing and verification algorithms can be accelerated by choosing $s = \log\log T$ and iterating the composition $\log T - \log\log\log T$ times, which lengthens signatures (that still keep sub-logarithmic size).

Note that we used the concrete forward-secure signature of Abdalla-Reyzin [2] (which itself extends [31]) but we could have utilized the Fiat-Shamir-based [18] scheme of Bellare-Miner in the same way as originally suggested in [12]. Unfortunately, it would significantly increase the size of signatures that would include long Fiat-Shamir public keys. The Kozlov-Reyzin system [26], where signatures also prove knowledge of the 2^{s+1-j}-root of some public element, can be hybridized in the same way to provide faster updates since a single modular squaring (instead of ℓ) suffices. In this case, signing is also faster since $O(t)$ squarings suffice to compute S'_t from S_t at the "key unblinding" of step 3.

Security proofs in the random oracle model under the factoring assumption are easily obtained using the forking lemma [32] combined with ideas from [2, 4].

THEOREM 4.3. *The above hybrid scheme has secure updates in the random oracle model assuming that factoring Blum integers is hard.*

PROOF. Detailed in the full version of the paper. □

4.2 The Product Composition

The sum composition is sufficient to obtain forward secure signatures with untrusted updates with T periods from any FSS-UU scheme with only 1 period. However, to benefit from the full power of the MMM construction (and notably obtain a key generation of constant cost), we also need a product composition like the one suggested in [28]. Recall that the latter combines two traditional forward-secure signatures with respectively T_0 and T_1 periods into a FSS scheme over $T_0 \cdot T_1$ stages. The idea is to use a new instance of the second scheme Σ_1 at each period (called *epoch*) of the first scheme Σ_0. At the beginning of each epoch, the newly generated public key for Σ_1 is certified by means of a signature from Σ_0. In the product system $\Sigma_0 \otimes \Sigma_1$, a signature for period t consists of a signature generated using the Σ_0 scheme at epoch $\lfloor t/T_1 \rfloor$ as well as a signature and a public key for the Σ_1 scheme at period $t \bmod T_1$.

The product composition is not directly adaptable to the untrusted update setting in that we cannot obtain a FSS-UU with $T_0 \cdot T_1$ periods by combining two such schemes with T_0 and T_1 periods. The difficulty is that the update algorithm of $\Sigma_0 \otimes \Sigma_1$ uses the signing algorithm of Σ_0 which needs the second factor DecK in an untrusted update setting. As a consequence, the second factor would be involved in the update algorithm of the product composition, which is what we want to avoid.

Nevertheless, this section shows that a FSS-UU scheme $\Sigma_1^{\mathsf{UU}} = (\mathsf{Keygen}_1, \mathsf{CheckKey}_1, \mathsf{Update}_1, \mathsf{Sign}_1, \mathsf{Verify}_1)$ resulting from the sum compostion can be combined with a regular (i.e. without untrusted updates) forward-secure signature $\Sigma_0 = (\mathsf{Keygen}_0, \mathsf{Update}_0, \mathsf{Sign}_0, \mathsf{Verify}_0)$ over T_0 stages so as to obtain a product scheme $\Sigma_0 \otimes \Sigma_1^{\mathsf{UU}}$ with $T_0 \cdot T_1$ periods. We insist that the result does not appear to be true in general. A necessary condition is to use a scheme Σ_1^{UU} resulting from the sum composition applied $\log T_1$ times. Moreover, the proof of update security demands that the underlying 2-party multi-signature be secure in the *plain public key model* of [7], as opposed to the relaxed registered public key model used in [9, 27]. Hence, it is not clear whether or not a

[2]To minimize signature sizes, long public keys of [27, 36] should be included in the public key of the FSS-UU scheme. In this case, the public key component to be included in each signature only consists of one group element.

product combination of the scheme in [12] with a regular forward-secure signature yields a FSS-UU scheme.

Since public keys of Σ_1^{UU} are certified using the signing algorithm of a traditional FSS scheme at each epoch, the second factor is not needed in the update algorithm of the product $\Sigma_0 \otimes \Sigma_1^{\mathsf{UU}}$.

4.2.1 Description

In the resulting product system, called FSS-UU$^{\otimes}$ hereafter, the key generation algorithm randomly selects a second factor DecK which is used in instances of Σ_1^{UU} throughout all epochs. This factor is utilized as a random seed to generate a key pair $(\tilde{sk}, \tilde{pk})$ for the 2-party multi-signature scheme $\mathsf{2MS}$ that serves as a building block to construct Σ_1^{UU}. Note that $\tilde{pk}$ and the public parameters cp of $\mathsf{2MS}$ are normally included in the public key of Σ_1^{UU}. However, in order to minimize private key and signature sizes and avoid unnecessary redundancies in the overall storage, we only include them as public key elements for FSS-UU$^{\otimes} = \Sigma_0 \otimes \Sigma_1^{\mathsf{UU}}$.

```
$\mathsf{Keygen}(\lambda, r, T)$
    Choose $T_0, T_1$ s.t. $T = T_0 \cdot T_1$;
    $\mathsf{cp} \leftarrow \mathsf{PG}^{\mathsf{MS}}(\lambda)$;
    $(r, \mathsf{DecK}) \leftarrow G(r)$;
    $(\tilde{sk}, \tilde{pk}) \leftarrow \mathsf{Kg}^{\mathsf{MS}}(\mathsf{DecK})$;
    $pk' \leftarrow (\mathsf{cp}, \tilde{pk})$;
    $(r_0, r_1) \leftarrow G(r)$;
    $(r_1', r_1'') \leftarrow G(r_1)$;
    $(\mathsf{SK}_0, \mathsf{PK}^{\mathsf{FS}}) \leftarrow \mathsf{Keygen}_0(\lambda, r_0, T_0)$;
    $(\mathsf{EncSK}_0, \mathsf{PK}^{\mathsf{UU}}) \leftarrow \mathsf{Keygen}_1(\lambda, r_1', T_1, pk')$;
    $\sigma_0 \leftarrow \mathsf{Sign}_0(0, T_0, \mathsf{SK}_0, \mathsf{PK}^{\mathsf{UU}})$;
    $\mathsf{SK}_0 \leftarrow \mathsf{Update}_0(0, T_0, \mathsf{SK}_0, \mathsf{PK}^{\mathsf{FS}})$;
    $\mathsf{PK} \leftarrow (pk', \mathsf{PK}^{\mathsf{FS}})$;
    Parse $\mathsf{PK}^{\mathsf{UU}}$ as $(pk', \mathsf{PK}')$;
                                   /* $\mathsf{PK}'$ is a hash value */
    Return $(\underbrace{\langle \mathsf{SK}_0, \sigma_0, \mathsf{EncSK}_0, \mathsf{PK}', r_1'' \rangle}_{\mathsf{EncSK}}, \mathsf{PK})$;

$\mathsf{CheckKey}\big(t, T, \overbrace{\langle \mathsf{SK}_0, \sigma_0, \mathsf{EncSK}', \mathsf{PK}', r \rangle}^{\mathsf{EncSK}}, \mathsf{PK}\big)$
    Parse $\mathsf{PK}$ as $(pk', \mathsf{PK}^{\mathsf{FS}})$ ; Set $\mathsf{PK}^{\mathsf{UU}} \leftarrow (pk', \mathsf{PK}')$;
    Output $\perp$ if $(\mathsf{SK}_0, \mathsf{PK}^{\mathsf{FS}})$ is not a valid
    key pair for $\Sigma_0$ for period $\lfloor t/T_1 \rfloor$ or if
    $\mathsf{Verify}_0(\lfloor t/T_1 \rfloor, T_0, \mathsf{PK}^{\mathsf{FS}}, \mathsf{PK}^{\mathsf{UU}}, \sigma_0) = \perp$;
    Return $\mathsf{CheckKey}_1\big(t \bmod T_1, T_1, \mathsf{EncSK}', \mathsf{PK}^{\mathsf{UU}}\big)$;

$\mathsf{Update}\big(t, T, \overbrace{\langle \mathsf{SK}_0, \sigma_0, \mathsf{EncSK}', \mathsf{PK}', r \rangle}^{\mathsf{EncSK}}, \mathsf{PK}\big)$
    If $(t = T - 1)$ erase $\mathsf{EncSK}$ and return “no period left”;
    Parse $\mathsf{PK}$ as $(pk', \mathsf{PK}^{\mathsf{FS}})$; Set $\mathsf{PK}^{\mathsf{UU}} \leftarrow (pk', \mathsf{PK}')$;
    If $(t + 1 \neq 0 \mod T_1)$
        $\mathsf{EncSK}' \leftarrow \mathsf{Update}_1\big(t \bmod T_1, T_1, \mathsf{EncSK}', \mathsf{PK}^{\mathsf{UU}}\big)$;
    else
                                   /* initialization of a new epoch */
        $(r', r) \leftarrow G(r)$;
        $\big(\mathsf{EncSK}', (pk', \mathsf{PK}')\big) \leftarrow \mathsf{Keygen}_1(\lambda, r', T_1, pk')$;
        $\sigma_0 \leftarrow \mathsf{Sign}_0\big(\lfloor t/T_1 \rfloor, T_0, \mathsf{SK}_0, (pk', \mathsf{PK}')\big)$;
        $\mathsf{SK}_0 \leftarrow \mathsf{Update}_0(\lfloor t/T_1 \rfloor, T_0, \mathsf{SK}_0, \mathsf{PK}^{\mathsf{FS}})$;
    endif

$\mathsf{Sign}\big(t, T, \overbrace{\langle \mathsf{SK}_0, \sigma_0, \mathsf{EncSK}', \mathsf{PK}', r \rangle}^{\mathsf{EncSK}}, \mathsf{DecK}, M, \mathsf{PK}\big)$
    Parse $\mathsf{PK}$ as $(pk', \mathsf{PK}^{\mathsf{FS}})$; Set $\mathsf{PK}^{\mathsf{UU}} \leftarrow (pk', \mathsf{PK}')$;
    $\sigma_1 \leftarrow \mathsf{Sign}_1(t \bmod T_1, T_1, \mathsf{EncSK}', \mathsf{DecK}, M||\mathsf{PK}'||t, \mathsf{PK}^{\mathsf{UU}})$
    Return $(\underbrace{\langle \mathsf{PK}', \sigma_0, \sigma_1 \rangle}_{\sigma}, t)$

$\mathsf{Verify}\big(t, T, \mathsf{PK}, M, \overbrace{\langle \mathsf{PK}', \sigma_0, \sigma_1 \rangle}^{\sigma}\big)$
    Parse $\mathsf{PK}$ as $(pk', \mathsf{PK}^{\mathsf{FS}})$; Set $\mathsf{PK}^{\mathsf{UU}} \leftarrow (pk', \mathsf{PK}')$;
    Return $\perp$ if $\mathsf{Verify}_0(\lfloor t/T_1 \rfloor, T_0, \mathsf{PK}^{\mathsf{FS}}, \mathsf{PK}^{\mathsf{UU}}, \sigma_0) = \perp$;
    Return $\perp$ if
        $\mathsf{Verify}_1(t \mod T_1, T_1, \mathsf{PK}^{\mathsf{UU}}, M||\mathsf{PK}'||t, \sigma_1) = \perp$;
    Return $\top$;
```

When the subroutine Sign_1 is called by the the signing algorithm, the public key PK' is included in the augmented message $M||\mathsf{PK}'||t$ in order to be consistent of the Bellare-Neven security model for multi-signatures. When restricted to bipartite signatures, the latter allows the adversary to produce her forgery on a previously signed message for a different second party of adversarially-chosen public key.

4.2.2 Security

When restricted to schemes involving two signers, the model of [7] considers a game where a challenger $\mathcal{C}$ generates a key pair $(\tilde{sk}, \tilde{pk})$ for a single honest signer. The public key $\tilde{pk}$ is given to the adversary $\mathcal{A}$. A polynomial number of times, the latter chooses arbitrary messages M and public keys pk to start a protocol instance with $\mathcal{C}$. The latter uses $\tilde{sk}$ carry out operations on behalf of the honest signer and interacts with $\mathcal{A}$ that plays the role of the corrupt party. An arbitrary number of concurrent instances of the protocol may be started. Upon termination, $\mathcal{A}$ outputs an arbitrary public key $pk^\star$ along with a message-signature pair (M, σ). She wins if σ is a valid signature on M for public keys $(pk^\star, \tilde{pk})$ and $\mathcal{C}$ was never involved in an execution of the protocol on message M with a co-signer of public key $pk^\star$. The strength of the model lies in that $\mathcal{A}$ is not required to hand over private keys matching adversarially-chosen public keys in signing queries or in the forgery stage.

THEOREM 4.4. *If the underlying 2-party multi-signature* $\mathsf{2MS}$ *is secure against chosen-message attacks in the plain public key model, FSS-UU*$^{\otimes}$ *ensures update security. Namely, an adversary* $\mathcal{F}$ *in the sense of definition 2 implies a chosen-message attacker* $\mathcal{A}$ *having identical advantage* $\mathsf{AdvUS}(\mathcal{F})$ *over* $\mathsf{2MS}$ *within comparable running time.*

PROOF. We outline a chosen-message attacker $\mathcal{A}$ against $\mathsf{2MS} = (\mathsf{PG}^{\mathsf{MS}}, \mathsf{Kg}^{\mathsf{MS}}, \mathsf{Sign}^{\mathsf{MS}}, \mathsf{Verify}^{\mathsf{MS}})$ using $\mathcal{F}$ as a subroutine and which is successful (in the plain public key model) whenever $\mathcal{F}$ is so. Algorithm $\mathcal{A}$ interacts with a challenger $\mathcal{C}$ that gives her a challenge public key $\tilde{pk}$ for $\mathsf{2MS}$.

At any time, $\mathcal{A}$ may start a joint execution of the signing protocol with $\mathcal{C}$ which plays the role of the honest signer $\tilde{pk}$ whilst she emulates the unregistered corrupt signer pk.

$\mathcal{F}$ gets as input a public key for FSS-UU$^{\otimes}$ over $T = T_0 \cdot T_1$ stages. This public key is produced almost exactly as in the key generation algorithm of FSS-UU$^{\otimes}$. Namely, $\mathcal{A}$ runs Keygen but defines $pk' = (\mathsf{cp}, \tilde{pk})$ using her own challenge public key $\tilde{pk}$ as well as the public parameters cp received from $\mathcal{C}$ instead of generating a pair $(\tilde{sk}, \tilde{pk})$ from a second factor. In other words, $\mathcal{A}$ skips lines 2 to 4 in the above description of Keygen and defines pk' using her own input at line 5. Since DecK is not involved in calls to Keygen_0, Keygen_1, Sign_0 or Update_0, remaining operations of Keygen are carried out as in the real game. The initial "encrypted" private key EncSK_0 for period 0 in FSS-UU$^{\otimes}$ is given to $\mathcal{F}$ who starts making queries.

Update queries: since the second factor DecK is not involved in update operations, $\mathcal{A}$ can perfectly answer update queries as in the proof of theorem 4.1.

Signing queries: at any period t, $\mathcal{F}$ may query $\mathcal{A}$ to sign a message M. To answer such a request, $\mathcal{A}$ can always compute σ_0 (the certificate for Σ_1^{UU} at epoch $\lfloor t/T_1 \rfloor$) herself since she knows the private key SK_0 of Σ_0 from the key generation stage. To obtain σ_1 (which is a signature on the augmented message $M||\mathsf{PK}'||t$), she triggers the recursive signing algorithm Sign_1 and follows its specification until entering the last step of the recursion (when $T = 1$). At this point, $\mathcal{A}$ must query her challenger $\mathcal{C}$ to obtain a multi-signature σ' w.r.t her challenge public key $\tilde{pk}$ and another public key pk for which she knows the matching secret sk that she chose herself (during the recursive key generation of an instance of FSS-UU$^{\oplus}$) without being required to reveal it when the update oracle was queried to enter epoch $\lfloor t/T_1 \rfloor$. This sub-key sk of EncSK' allows her to play the malicious party in her interaction with $\mathcal{C}$. The resulting signature σ' completes the recursion and allows her obtaining (σ_1, PK'), which finishes the signature generation.

Forgery: eventually, $\mathcal{F}$ is expected to produce a forgery $\sigma^\star = (\langle \mathsf{PK}'^\star, \sigma_0^\star, \sigma_1^\star \rangle, t^\star)$ on a message $M^\star$ for some period $t^\star$ during which $M^\star$ was not queried for signature.

As in the security proof of the original product composition (theorem 4 in [28]), we have to distinguish two cases. Let $\mathcal{SEEN}$ denote the set $\{\mathsf{PK}'_1, \ldots, \mathsf{PK}'_e\}$ of public key components for Σ_1^{UU} that $\mathcal{F}$ happens to observe within outputs of signing queries. If $\mathsf{PK}'^\star \notin \mathcal{SEEN}$, the inner multi-signature of $\sigma_1^\star$ (which consists of a 2-party signature, a public key $\overline{pk}$ and $O(\log T_1)$ hash values) is a 2-party signature on a new message $M^\star||\mathsf{PK}'^\star||t^\star||t^\star \bmod T_1$ w.r.t. the challenge public key $\tilde{pk}$ and some other public key $\overline{pk}$ for which $\mathcal{A}$ may not know the corresponding secret $\overline{sk}$. However, this suffices to break the unforgeability of $\mathsf{2MS}$ in the plain public key model (where $\mathcal{A}$ is not required to know or reveal the private keys of maliciously generated public keys).

On the other hand, if $\mathsf{PK}'^\star \in \mathcal{SEEN}$, $\sigma_1^\star$ necessarily contains a multi-signature on message $M^\star||\mathsf{PK}'^\star||t^\star||t^\star \bmod T_1$ (that was not previously queried since the pair $(M^\star, t^\star)$ was not involved in a signing query from $\mathcal{F}$) w.r.t. public keys $\tilde{pk}$ and a public key pk of known secret key sk, which also implies a breach in the security of $\mathsf{2MS}$. $\square$

Theorem 4.5. *Assuming the security of the underlying* $\mathsf{2MS}$ *scheme in the plain public key model and the forward-security of* Σ_0 *in the sense of Bellare-Miner, the FSS-UU*$^{\otimes}$ *composition is forward-secure in the sense of definition 1. Namely, for a product scheme over* $T_0 \cdot T_1$ *periods, a forward security adversary* $\mathcal{F}$ *has at most advantage*

$$\mathsf{AdvFS}(\mathcal{F}) \leq \mathsf{AdvFS}(\mathcal{F}_0) + T_0 \cdot T_1 \cdot \mathsf{AdvMS}(\mathcal{F}^{\mathsf{MS}})$$

after q_s *and* q_u *signing and update queries within time*

$$t' \leq \max\{t_0 - q_s \cdot t_{\mathsf{Sng}}^{\mathsf{MS}} - T_0 \cdot T_1 \cdot t_{\mathsf{Kg}}^{\mathsf{MS}}, \quad t_1 - q_s \cdot t_{\mathsf{Sng}}^{\mathsf{MS}} - T_0 \cdot (T_1 \cdot t_{\mathsf{Kg}}^{\mathsf{MS}} + t_{\mathsf{Sgn}}^{\mathsf{FS}})\},$$

where $t_{\mathsf{Sgn}}^{\mathsf{FS}}$ *and* $t_{\mathsf{Kg}}^{\mathsf{FS}}$ *respectively denote the time complexities of signing and key generation algorithms in* Σ_0, $t_{\mathsf{Sng}}^{\mathsf{MS}}$ *and* $t_{\mathsf{Kg}}^{\mathsf{MS}}$ *stand for these costs in* $\mathsf{2MS}$ *while* $\mathsf{AdvFS}(\mathcal{F}_0)$ *and* $\mathsf{AdvMS}(\mathcal{F}^{\mathsf{MS}})$ *denote maximal advantages of a forward security adversary* $\mathcal{F}_0$ *against* Σ_0 *and a forger* $\mathcal{F}^{\mathsf{MS}}$ *against the* $\mathsf{2MS}$ *scheme.*

Proof. The result stems from the proof of theorem 4 in [28]. As in theorem 4.2, it suffices to observe that FSS-UU schemes become traditional forward-secure signatures when the adversary knows DecK as in definition 1 and that FSS-UU systems with one period can be implemented by 2-party multi-signatures in the plain public key model. $\square$

4.3 Extending MMM

Recall that Malkin *et al.* [28] generically obtain forward-secure signatures from any digital signatures by suitably integrating their sum and product compositions.

The salient property of the construction is that it does not require to know the number of time period at key generation time and allows for schemes with (virtually) unbounded lifetime: the only theoretical bound on the number of periods is exponential in security parameters of underlying symmetric primitives (i.e. a pseudorandom generator and a collision-resistant hash function) and thus essentially impossible to reach in practice. In all metrics, the MMM scheme never exceeds a complexity that mildly (i.e. logarithmically) depends on the number of periods elapsed so far.

In a nutshell, the construction is a product composition $\Sigma_0 \otimes \Sigma_1$ where epochs use instances of a FSS scheme Σ_1 with increasingly large numbers of periods, which is what allows for a complexity growing as time elapses instead of depending on a maximal number of stages. During epoch j the product scheme uses an instance of Σ_1 with 2^j periods (obtained by j iterations of the sum composition). If ℓ is the security parameter of underlying symmetric primitives, the product involves ℓ epochs (i.e. a scheme Σ_0 with ℓ periods resulting from the sum composition applied $\log \ell$ times) so that the theoretical overall number of stages $\sum_{j=0}^{\ell-1} 2^j = 2^\ell - 1$ is far beyond the needs of any practical application.

From an efficiency point of view, resulting signatures at time period t consist of only two digital signatures, two public keys and $\log \ell + \log t$ hash values (more precisely, $\log \ell$ of them stem from the sum composition producing Σ_0 and remaining $\log t$ hash values pertain to a second sum composition at epoch $j = O(\log t)$ of the product). Signature generation only requires to compute a digital signature and verification entails the verification of 2 digital signatures as well as $\log \ell + \log t$ hash operations. Public keys only consist of a hash value while private keys logarithmically grow as time goes by (their length is $O(\lambda + (\log \ell + \log t)\ell)$ bits). When amortized (we refer to [28] for more details), the cost of an update operation at period t is given by $O(\lambda^2 \ell + \ell^2 \log t)$ and the complexity of the key generation algorithm only depends on security parameters λ and ℓ.

By integrating our modified sum and product compositions of sections 4.1 and 4.2 in the same way, we can obtain a forward-secure signatures with untrusted updates enjoying identical performance. We first construct a regular FSS scheme Σ_0 with ℓ periods thanks to the original sum composition [28]. Then, we obtain a "twisted product" by using an instance of Σ_1^{UU} with 2^j periods at epoch j. Each instance of Σ_1^{UU} should result from applying FSS-UU$^{\oplus}$ to any 2-party multi-signature in the plain public key model. The Schnorr-based [34] construction of [7] is a good candidate and so are its alternative implementations based on RSA [20], factoring [18, 31] or the Decision Diffie-Hellman assumption [24]. As mentioned in [7], unrestricted aggregate signatures put forth in [6] also give rise to multi-signatures in the plain public key model that can be used here as well.

Again, several tradeoffs are possible. For instance, the regular FSS scheme Σ_0 in our "twisted product" can be a number theoretic signature such as the one of Itkis-Reyzin [22] instantiated over ℓ periods (recall that ℓ is the security parameter of a symmetric primitive and is thus relatively small w.r.t. realistic numbers of periods T). This removes the need for including $\log \ell$ hash values in signatures while linear key generation and updates from the first version of [22] are avoided. Of course, the same idea applies to the original MMM system as well.

Our full construction currently applies to only a handful of schemes. Also, the only known examples [7, 6] of multi-signatures in the plain public key model rely on the random oracle methodology [5]. To date, it turns out that we can only take full advantage of the MMM construction with random-oracle-using signatures. However, security proofs of our modified sum and product compositions do not rely on random oracles. We thus believe that our extension of MMM is an additional incentive to seek after standard model realizations of multi-signatures in the plain public key model.

5. CONCLUSION

In this paper, we described new constructions of forward-secure signature with the untrusted update property recently put forth in [12]. Our generic construction from any forward-secure signature is very simple but induces size and computational overheads. By extending the very efficient MMM sum-product composition however, we obtain a number of schemes based on various - non pairing-related - computational assumptions and featuring very attractive performance. This resolves an open problem raised in [12] that called for efficient implementations of untrusted updates in existing forward-secure signatures found in the literature.

When applied to the recently suggested multi-signatures of Bellare-Neven [7], our extension of MMM notably provides FSS-UU schemes with a practically unbounded number of time periods. It does not introduce additional random oracle assumptions either. Currently known instantiations of these new "unbounded" systems rely on random oracles only because the underlying multi-signatures do.

6. REFERENCES

[1] M. Abdalla, S. K. Miner, C. Namprempre. Forward-Secure Threshold Signature Schemes. In *CT-RSA*, pp. 441–456, 2001.

[2] M. Abdalla, L. Reyzin. A New Forward-Secure Digital Signature Scheme. In *ASIACRYPT*, pp. 116–129, 2000.

[3] R. Anderson. Two Remarks on Public Key Cryptology. Invited lecture, *ACM CCS*, 1997.

[4] M. Bellare, S. Miner. A Forward-Secure Digital Signature Scheme. In *CRYPTO*, pp. 431–448, 1999.

[5] M. Bellare, P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS*, pp. 62–73, 1993.

[6] M. Bellare, C. Namprempre, G. Neven. Unrestricted Aggregate Signatures. In *ICALP*, pp. 411–422, 2007.

[7] M. Bellare, G. Neven. Multi-signatures in the plain public-Key model and a general forking lemma. In *ACM CCS*, pp. 390–399, 2006.

[8] M. Bellare, B. Yee. Forward-Security in Private-Key Cryptography. In *CT-RSA*, pp. 1–18, 2003.

[9] A. Boldyreva. Efficient Threshold Signature, Multisignature and Blind Signature Schemes Based on the Gap-Diffie-Hellman-group Signature Scheme. In *PKC*, pp. 31–46, 2003.

[10] D. Boneh, X. Boyen, E.-J. Goh. Hierarchical Identity Based Encryption with Constant Size Ciphertext. In *EUROCRYPT*, pp. 440–456, 2005.

[11] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In *ASIACRYPT*, pp. 514–532, 2001.

[12] X. Boyen, H. Shacham, E. Shen, B. Waters. Forward-Secure Signatures with Untrusted Update. In *ACM CCS*, pp. 191–200, 2006.

[13] R. Canetti, S. Halevi, J. Katz. A Forward Secure Public Key Encryption Scheme. In *EUROCRYPT*, pp. 254–271, 2003.

[14] E. Cronin, S. Jamin, T. Malkin, P. McDaniel. On the Performance, Feasibility and Use of Forward-Secure Signatures. In *ACM CCS*, pp. 131–144, 2003.

[15] Y. Dodis, J. Katz, S. Xu, M. Yung. Key-Insulated Public Key Cryptosystems. In *EUROCRYPT'02*, pp. 65–82, 2002.

[16] Y. Dodis, J. Katz, S. Xu, M. Yung. Strong key-insulated signature schemes. In *PKC*, pp. 130–144, 2003.

[17] Y. Dodis, L. Reyzin. Breaking and repairing optimistic fair exchange from PODC 2003. In Digital Rights Management Workshop 2003, pp. 47–54, 2003.

[18] A. Fiat and A. Shamir. How to prove yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO*, pp. 186–194, 1986.

[19] S. Goldwasser, S. Micali, R. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM J. Comput.* 17(2), pp. 281–308, 1988.

[20] L. Guillou, J.-J. Quisquater. A "paradoxical" identity-based signature scheme resulting from zero-knowledge. In *CRYPTO*, pp. 216–231, 1988.

[21] K. Itakura, K. Nakamura. A Public Key Cryptosystem Suitable for Digital Multisignatures. In *NEC Research & Development*, 71:1–8, 1983.

[22] G. Itkis, L. Reyzin. Forward-Secure Signatures with Optimal Signing and Verifying. In *CRYPTO*, pp. 332–354, 2001.

[23] G. Itkis, L. Reyzin. SiBIR: Signer-Base Intrusion-Resilient Signatures. In *CRYPTO*, pp. 499–514, 2002.

[24] J. Katz, N. Wang. Efficiency Improvements for Signature Schemes with Tight Security Reductions. In *ACM CCS*, pp. 155–164, 2003.

[25] H. Krawczyk. Simple Forward-Secure Signatures from any Signature Scheme. *ACM CCS*, pp. 108-115, 2000.

[26] A. Kozlov, L. Reyzin. Forward-Secure Signatures with Fast Key Update. In *SCN*, pp. 241–256, 2002.

[27] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, B. Waters. Sequential Aggregate Signatures and Multisignatures Without Random Oracles. In *EUROCRYPT*, pp. 465–485, 2006.

[28] T. Malkin, D. Micciancio, S. K. Miner. Efficient Generic Forward-Secure Signatures with an Unbounded Number Of Time Periods. In *EUROCRYPT*, pp. 400–417, 2002.

[29] R. Merkle. A Digital Signature Based on a Conventional Encryption Function. In *CRYPTO*, pp. 369–378, 1988.

[30] S. Micali, K. Ohta, L. Reyzin. Accountable-subgroup multisignatures. *ACM CCS*, pp. 245–254, 2001.

[31] H. Ong and C. P. Schnorr. Fast signature generation with a Fiat-Shamir like scheme. In *EUROCRYPT*, pp. 432–440, 1990.

[32] D. Pointcheval and J. Stern. Security proofs for signature schemes. In *EUROCRYPT*, pp. 387–398, 1996.

[33] J. Rompel. One-Way Functions are Necessary and Sufficient for Secure Signatures. In *STOC*, pp. 387–394, 1990.

[34] C. P. Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO*, pp. 239–252, 1989.

[35] D. Song. Practical Forward-Secure Group Signature Schemes. *ACM CCS*, pp. 225–234, 2001.

[36] B. Waters. Efficient Identity-Based Encryption Without Random Oracles. In *EUROCRYPT*, pp. 114–127, 2005.