Multi-Signatures in the Plain Public-Key Model and a General Forking Lemma

Mihir Bellare
University of California San Diego
Department of Computer Science & Engineering
9500 Gilman Drive
La Jolla, California 92093, USA
mihir@cs.ucsd.edu

Gregory Neven
Katholieke Universiteit Leuven
B-3001 Heverlee, Belgium;
and Ecole Normale Supérieure
75005 Paris, France
Gregory.Neven@esat.kuleuven.be

ABSTRACT

A multi-signature scheme enables a group of signers to produce a compact, joint signature on a common document, and has many potential uses. However, existing schemes impose key setup or PKI requirements that make them impractical, such as requiring a dedicated, distributed key generation protocol amongst potential signers, or assuming strong, concurrent zero-knowledge proofs of knowledge of secret keys done to the CA at key registration. These requirements limit the use of the schemes. We provide a new scheme that is proven secure in the plain public-key model, meaning requires nothing more than that each signer has a (certified) public key. Furthermore, the important simplification in key management achieved is not at the cost of efficiency or assurance: our scheme matches or surpasses known ones in terms of signing time, verification time and signature size, and is proven secure in the random-oracle model under a standard (not bilinear map related) assumption. The proof is based on a simplified and general Forking Lemma that may be of independent interest.

Categories and Subject Descriptors

C.2.0 [**General**]: Security and protection

General Terms

Security

Keywords

Cryptography, digital signatures, multi-signatures, Forking Lemma.

1. INTRODUCTION

Consider entities $1, \ldots, N$, each having a public key and corresponding secret key. A multi-signature (MS) scheme allows any subset $L \subseteq \{1, \ldots, N\}$ of them, at any time, to engage in an interactive protocol whose output is a joint signature on a message m of their choice. Verification can be done by any external party given just L, m, the purported multi-signature σ, and the public keys of all signers in L. Such a system could be useful for contract signing, co-signing, or distribution of a certificate authority.

A trivial way to implement a multi-signature scheme is to let the multi-signature σ of message m be the list $(\sigma_i : i \in L)$ where σ_i is i's signature on m. This multi-signature is however large, in particular of size proportional to the number $|L|$ of signers. There are several important practical reasons for which this is costly and undesirable. For example, on wireless devices such as PDAs, cell phones, RFID chips and sensors, battery life is the main limitation. Communicating even one bit of data uses significantly more power than executing one 32-bit instruction [4]. Reducing the number of bits to communicate saves power and is important to increase battery life. Also, in many settings, communication is not reliable, and so the fewer the number of bits one has to communicate, the better. For such reasons, we want multi-signature schemes that are non-trivial, meaning the size of the multi-signature is about the same as that of a single ordinary signature and in particular not proportional to the number $|L|$ of signers.

Rogue-key attacks. The early literature of MS schemes [25, 20, 28, 24, 27, 32, 34, 35] features numerous attacks breaking proposed schemes. In most cases, this was due to weaknesses related to key setup, in particular the ability to mount a *rogue-key attack*. In such an attack, an adversary who is a group member (insider) chooses its public key as a function of that of honest users in such a way that it can then easily forge multi-signatures. Although they might at first hearing sound far-fetched, rogue-key attacks are in fact possible to mount in practice and are a real threat. When, eventually, precise definitions [31] and proven secure schemes [31, 11, 29] emerged, they obviously paid a lot of attention to key setup. These schemes were, happily, proven secure against rogue-key attacks, but, unhappily, at the cost of complexity and expense in the scheme, or using unrealistic and burdensome assumptions on the public-key infrastructure (PKI), as we will now explain in detail.

Drawbacks of previous schemes. The MS-MOR [31] scheme requires, as a pre-processing step, that the set of *potential* signers engages in an interactive key generation protocol that provides to each a public and secret key. The purpose of the protocol is to ensure that no dishonest player can choose its public key as a function of public keys of honest players. This type of *dedicated key generation* is however not practical for several reasons. First, it means the group of potential signers is restricted to being static: it must be decided and fixed before signing can start, and new signers cannot be added later. However, in practice the potential set of signers is dynamic and may not be known ahead of time, and we would like to be able to add potential signers at will. Second, the key generation protocol of [31] is expensive and results in large, complex public keys. In particular, the public key of each signer has a size that depends on the number N of potential signers. (However, once keys are established, multi-signature generation and verification are actually attractively cheap.)

The drawback of the MS-Bo [11] and MS-LOSSW [29] MS schemes is that the security model makes the *knowledge of secret key* (KOSK) assumption. There is no dedicated key generation, but, when the adversary, mounting a rogue-key attack, provides a public key of its choice for a group member, it is required, in the model, to provide also a matching secret key. Of course "real" adversaries would not do any such thing, so what does this mean? It is explained by the authors as modeling the assumption that a user provides the certification authority (CA) with a proof of knowledge of its secret key before the CA certifies the corresponding public key. However, it is not that simple. The current implementation of these proofs is represented by standards PKCS#10 [38] —used by VeriSign— and RFC 4210/11 [3, 41]. Here the proof is implemented by having the user send the CA a signature, under the public key it is attempting to get certified, of some message that includes the public key and the user identity. While such methods might intuitively seem to prove knowledge of the secret key, they do *not* suffice to realize the abstract model of [11, 29] in which the attacker actually hands the challenger the secret keys. In particular, not only does this type of proof of possession not suffice to prove secure the MS-Bo and MS-LOSSW schemes, but there are actually attacks against these schemes if such proofs of possession are used [9]. (That is, if we attempt to drop the KOSK assumption and substitute these proofs of possession instead.) To obtain proofs of possession sufficient to implement the KOSK assumption, it appears one should use zero-knowledge (ZK) proofs of knowledge (POKs) that meet strong, formal extractability requirements [5]. However, that is not all. Since in practice we would expect that the users may register keys at any time, concurrently with other users or with executions of the multi-signature protocol, one would require ZK POKs extractable under such concurrent conditions. This eliminates many standard protocols, including standard POKs of discrete logarithms. Still, it is true that protocols with the desired properties do seem to exist. For example, in the random-oracle model one could use [17] or, in the standard model, non-interactive ZK POKs [14]. But the first is not cheap and the second is prohibitive, and even if one were willing, adding them to the PKI requires modifying the client and CA functioning and software, which is difficult, costly and preferably avoided. Also, one is still left with the task of actually formally justifying a claim that this would implement the abstract KOSK model of [11, 29]. (We are not making such a claim here.) However, the main reason this route is impractical is simply that CAs do not right now implement such POKs. If, today, some corporation or person wishes to implement a MS scheme, it seems unlikely that VeriSign is going to oblige them by suddenly changing their offerings to include appropriate POKs of secret keys at registration.

Our MS scheme. In summary, the most significant practical obstacle to multi-signatures at present is the key-setup requirements or assumptions of previous works. Our contribution is to remove this obstacle by presenting a multi-signature scheme in the *plain public-key model.* This means that, with regard to key setup, nothing more is required than in *any* usage of public-key cryptography, namely that any potential signer has a public key. There is no dedicated key generation protocol. A signer is *not* assumed to have proved knowledge of its secret key to the CA, but only to have a standard certificate. Yet, security against rogue-key attacks is proved *without* the KOSK assumption.

To elaborate, in our setting, the group of potential signers is dynamic: anyone possessing a (certified) public key can join at any time. In our security model, the adversary can corrupt a signer and choose its public key as a function of those of other (honest) signers. It is not required to supply the challenger with a matching secret key, meaning we prove security even when the adversary does not know the secret key underlying a public key it makes for itself. The fact that we do not need to assume any kind of proof-of-knowledge of the secret key performed to the CA at the time a public key is registered and a certificate is obtained reduces the demands on the PKI and allows our protocols to be implemented within the current PKI. CAs need not take any special actions or change their functioning or software. Indeed, a CA does not need to even know that a key is to be used in our multi-signature scheme; it can be treated like any other key.

Efficiency and other scheme attributes. The significant gains in key setup achieved by our MS scheme are not at the cost of performance. As Table 1 indicates, our scheme, denoted MS-BN, compares favorably with previous ones in terms of signing time, verifying time, signature size and other attributes. We now discuss the information in the table in a little more depth.

MS-BN is based on the Schnorr [42] scheme and is proven secure in the random-oracle (RO) model [10] assuming hardness of the standard discrete logarithm problem. (MS-Bo and MS-MOR also use the RO model, but MS-LOSSW does not.) MS-BN allows secure concurrent executions of signing protocols by different subsets of the set of potential signers, which is important because applications on the Internet are inherently placed in a concurrent execution environment. Concurrent signing is explicitly disallowed in [31]. Security of our scheme is proved even when the adversary can control the scheduling and mount rogue-key attacks, yet without the KOSK assumption.

Unlike [11, 29], we do not use pairings (bilinear maps), which not only results, for us, in greater efficiency and ease of implementation, but also means we do not rely on the relatively new and untested hardness assumptions related to pairing-based cryptography. Verification in MS-BN is cheaper than in MS-Bo, and both signing and verification are cheaper in MS-BN than in MS-LOSSW. We have included the system parameter size in the table mainly to note that this is very large (25,920 bits) for MS-LOSSW, unlike for any other scheme. Our signatures are 320 bits as opposed to the 160 of MS-Bo because the latter uses the

Scheme	Sign	Verify	|sig|	|pk|	|par|	Key setup	Assump
MS-MOR [31]	1 exp	1 exp	$2 \cdot 160$	$[3 + 2\lg(N)] \cdot 160$	0	dedicated key-reg	DL
MS-Bo [11]	1 exp	2 pr	160	$6 \cdot 160$	$6 \cdot 160$	KOSK model	(co)CDH
MS-LOSSW [29]	3 exp	2 pr	$7 \cdot 160$	$6 \cdot 160$	$162 \cdot 160$	KOSK model	(co)CDH
MS-BN	1 exp	1 exp	$2 \cdot 160$	160	160	plain pk model	DL

Table 1: MS scheme comparisons. For each scheme (the last is ours) we show the computational cost of signing (per signer), the computational cost of verification of a multi-signature, the size of a multi-signature, the size of the public key of an individual signer, the size of the system parameters common to all signers, the type of key-setup, and the assumption used to prove security. All sizes are in bits. By "exp" we mean an exponentiation. (Some of the exponentiations are actually multi-exponentiations, but these have the same cost as single exponentiations.) N is the total number of signers in the system. By "pr" we mean a pairing, whose cost estimate is 6–20 exponentiations. We assume we work over a 160-bit elliptic-curve (EC) group for the DL-based schemes. For the coCDH-based schemes we assume an asymmetric pairing, that is, e: $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with $\mathbb{G}_1 \neq \mathbb{G}_2$ (this to make the signatures as short as possible) and an isomorphism ψ: $\mathbb{G}_2 \to \mathbb{G}_1$ (this to make the proofs go through) with group-element representation sizes in $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ being, respectively, 160-bits, $6 \cdot 160$ bits and $6 \cdot 160$ bits, which is what is needed, in this asymmetric-with-isomorphism setting, to provide the 1024-bit RSA level of security achieved by 160-bit EC groups [18].

pairing-based BSL [13] short signature scheme, but our gain in verification time (by a factor of 12-40) more than compensates. The public keys in our scheme are shorter than in any other scheme, an important benefit in case they have to be transmitted with the multi-signature. Additionally, of course, MS-BN is in the plain public-key model. Our efficiency estimates do not take security into account, meaning that all schemes are not necessarily compared at the same level of security. We do this as we do not think our analyses are tight (that is, the real security is better) and thus comparing at the same level would be misleading for practice.

New Forking Lemma. Our proof of security of the MS-BN scheme relies on a generalization of the Forking Lemma of [40] that may be of independent interest. The original Forking Lemma of Pointcheval and Stern [40] applies to signature schemes obtained from three-move identification schemes via the Fiat-Shamir [16] transform in the random oracle model. Roughly it says that in an *expected* $O(1/\epsilon)$ repeated executions of a forger A with success probability ϵ, one can find two accepting conversations that agree in the first prover move but not the verifier challenge, leading, via the special soundness property of Σ protocols, to recovery of the secret key and hence a proof of security of the signature scheme in the RO model. This lemma has been important in proving security of signature schemes via the rewinding technique. However, the lemma seems hard (if not impossible) to apply in situations like ours where we are not dealing with a regular signature scheme but a MS scheme. Indeed, in the past, variants of the lemma had to be formulated and proved for different types of signatures. (For example, a version for blind signatures is in [40], and one for ring signatures in [22]. Another variant is in [39].) The statement of our Forking Lemma (cf. Lemma 1), in contrast to previous ones, makes no mention of signatures or even, for that matter, random oracles. Rather it asserts a simple lower bound on the probability that two executions of an arbitrary algorithm on certain (related) inputs both accept. This statement, we feel, distills the probabilistic essence of the Forking Lemma and divorces it from any particular application context. (In our view, the Forking Lemma is something purely probabilistic, not about signatures. Previous Forking Lemmas mixed these things up.) In this form, it can be be applied not only to prove security of regular signature schemes but also, as we show, to prove security of schemes like ours where the setting is more complex. Our Forking Lemma also provides worst-case rather than expected-time guarantees on the constructed algorithm, in contrast to [40]. We feel this meshes better with standard assumptions. (In using the Lemma of [40], you need to assume, say, hardness of discrete-logarithm computations against expected-time adversaries. This assumption may be true, but is not the standard one.) Our Forking Lemma can be viewed as an extension of the Reset Lemma of [8], and our proof, which uses the techniques of the latter, is simpler than that of [40].

Relation to aggregate signatures. A natural thought is that multi-signatures are a special case of aggregate signatures, and we know the latter have been implemented without the KOSK assumption [12], so doesn't this yield multi-signatures in the plain public-key model? Let us explore this.

Suppose signer i has produced a BSL signature [13] σ_i on a message m_i $(i = 1, \ldots, n)$. The procedure of [12] aggregates $\sigma_1, \ldots, \sigma_n$ into a single, aggregate signature σ. Multi-signatures is simply the special case where $m_i = m$ is a common message for all $i \in L$. Now, [12] do prove security without the KOSK assumption, but for this *need to assume that the messages $m_1, \ldots, m_n$ are distinct.* So the multi-signature case is exactly the one where they do *not* have security without the KOSK assumption. In fact, in this case, there actually is a rogue-key attack on the scheme. Indeed, this MS scheme is exactly MS-Bo, which we know is not secure against rogue-key attack without the KOSK assumption.

However, [12] also suggest a workaround to the message distinctness assumption. Have each signer prepend its public key to its message before signing, so that the individual signatures now are (in the MS case) on the *enhanced messages* $pk_1 \| m, \ldots, pk_l \| m$. Now, security is guaranteed as long as the enhanced messages are distinct. However, this is not enough for security against rogue key attack in our plain public-key model. If an attacker, playing the role of signer 2, sets its public key pk_2 to equal the public key pk_1 of honest user 1 (an easy task) and outputs some forgery on some message m for some group including signers $1, 2$, then we have a situation where two enhanced messages are the same, so the result of [12] does not apply. To fix this one can use the analysis of [6] that shows the scheme is secure even if enhanced messages are not distinct, and then we do obtain a secure MS scheme. However, verification of a multi-signature for n signers costs $n+1$ pairings, making it substantially less efficient than all the other schemes we have discussed, where verification time does not depend on the number of signers in the group.

Another potential route to multi-signatures is via sequential aggregate signatures [30]. These can be built from trapdoor permutation families in which there is a single domain underlying the entire family [30], but in fact there seems to be no example of such a family. RSA does not have the desired property. The authors build some RSA-based schemes directly, another one can be constructed using techniques from [21]. Some limitations of the schemes of [30] are lifted in [6]. However, a sequential-aggregate based MS scheme will require a number of communication rounds proportional to the number n of signers involved in a signature, as well as n applications of the trapdoor function to verify, while all previous protocols, including ours, are constant-round and have constant verification cost.

2. NOTATION

Let $\mathbb{N} = \{1, 2, 3, \ldots\}$. A string means a binary one. The empty string is denoted ε. If x, y are strings, then $|x|$ is the length of x. If $x_1, x_2, \ldots$ are objects then $x_1 \| x_2 \| \ldots$ denotes an encoding of them as strings from which the constituent objects are easily recoverable. If S is a (multi)set, then $|S|$ is its cardinality, $s \stackrel{\$}{\leftarrow} S$ denotes the operation of assigning to s an element of S chosen at random, and $\langle S \rangle$ is a unique encoding of S as a string. If A is a randomized algorithm, then $\mathsf{A}(x_1, \ldots; \rho)$ denotes its output on inputs $x_1, \ldots$ and coins ρ, while $y \stackrel{\$}{\leftarrow} \mathsf{A}(x_1, \ldots)$ means that we choose ρ at random and let $y = \mathsf{A}(x_1, \ldots; \rho)$.

3. A GENERAL FORKING LEMMA

Here we state and prove our Forking Lemma here that we will use later to prove the security of our multi-signature scheme. Our Forking Lemma, unlike that of Pointcheval and Stern [40], makes no mention of signature schemes or random oracles, but rather concentrates on the output behavior of an algorithm when run twice on related inputs. This makes it easily applicable in contexts other than standard signature schemes, and separates the probabilistic analysis of the rewinding from the actual simulation in the security proof, allowing for more modular (and hence easier to verify) proofs. In the following, think of x as a public key and $h_1, \ldots, h_q$ as replies to queries to a random oracle.

LEMMA 1. **[General Forking Lemma]** *Fix an integer $q \geq 1$ and a set H of size $h \geq 2$. Let A be a randomized algorithm that on input $x, h_1, \ldots, h_q$ returns a pair, the first element of which is an integer in the range $0, \ldots, q$ and the second element of which we refer to as a* side output. *Let* IG *be a randomized algorithm that we call the input generator. The accepting probability of A, denoted* acc*, is defined as the probability that $J \geq 1$ in the experiment*

$$x \stackrel{\$}{\leftarrow} \mathsf{IG}\ ;\ h_1, \ldots, h_q \stackrel{\$}{\leftarrow} H\ ;\ (J, \sigma) \stackrel{\$}{\leftarrow} \mathsf{A}(x, h_1, \ldots, h_q)\ .$$

The forking algorithm $\mathsf{F_A}$ *associated to* A *is the randomized algorithm that takes input x proceeds as follows:*

```
Algorithm $\mathsf{F_A}(x)$
    Pick coins $\rho$ for $\mathsf{A}$ at random
    $h_1, \ldots, h_q \stackrel{\$}{\leftarrow} H$
    $(I, \sigma) \leftarrow \mathsf{A}(x, h_1, \ldots, h_q; \rho)$
    If $I = 0$ then return $(0, \varepsilon, \varepsilon)$
    $h'_I, \ldots, h'_q \stackrel{\$}{\leftarrow} H$
    $(I', \sigma') \leftarrow \mathsf{A}(x, h_1, \ldots, h_{I-1}, h'_I, \ldots, h'_q; \rho)$
    If $(I = I'$ and $h_I \neq h'_I)$ then return $(1, \sigma, \sigma')$
    Else return $(0, \varepsilon, \varepsilon)$.
```

Let

$$\mathrm{frk} \;=\; \Pr\Big[\, b = 1 \;:\; x \stackrel{\$}{\leftarrow} \mathsf{IG}\ ;\ (b, \sigma, \sigma') \stackrel{\$}{\leftarrow} \mathsf{F_A}(x) \,\Big]\ .$$

Then

$$\mathrm{frk} \;\geq\; \mathrm{acc} \cdot \left(\frac{\mathrm{acc}}{q} - \frac{1}{h} \right)\ . \qquad (1)$$

Alternatively,

$$\mathrm{acc} \;\leq\; \frac{q}{h} + \sqrt{q \cdot \mathrm{frk}}\ . \quad \blacksquare \qquad (2)$$

We proceed to the proof. We first recall two sublemmas that we will use in the proof. The proofs are skipped here but provided in the full version of our paper [7] for completeness. The first is a standard fact which one can derive from Jensen's inequality, or as a consequence of the fact that the variance of any random variable is non-negative:

LEMMA 2. *Let X be real-valued random variable. Then* $\mathbf{E}\left[X^2\right] \geq \mathbf{E}\left[X\right]^2$. $\blacksquare$

The next lemma is actually a consequence of the above although it appears in [2] with a different proof:

LEMMA 3. *Suppose $q \geq 1$ is an integer, and $x_1, \ldots, x_q \geq 0$ are real numbers. Then*

$$\sum_{i=1}^{q} x_i^2 \;\geq\; \frac{1}{q} \cdot \left(\sum_{i=1}^{q} x_i \right)^2\ . \;\blacksquare$$

PROOF OF LEMMA 1. We first prove (1) and then show that it implies (2). For any input x let $\mathrm{acc}(x)$ denote the probability that $J \geq 1$ in the experiment

$$h_1, \ldots, h_q \stackrel{\$}{\leftarrow} H\ ;\ (J, \sigma) \stackrel{\$}{\leftarrow} \mathsf{A}(x, h_1, \ldots, h_q)\ .$$

Also let

$$\mathrm{frk}(x) \;=\; \Pr\Big[\, b = 1 \;:\; (b, \sigma, \sigma') \stackrel{\$}{\leftarrow} \mathsf{F_A}(x) \,\Big]\ .$$

We claim that for all x,

$$\mathrm{frk}(x) \;\geq\; \mathrm{acc}(x) \cdot \left(\frac{\mathrm{acc}(x)}{q} - \frac{1}{h} \right)\ . \qquad (3)$$

Then, with the expectation taken over $x \stackrel{\$}{\leftarrow} \mathsf{IG}$, we have

$$\begin{aligned}
\mathrm{frk} &= \mathbf{E}\left[\mathrm{frk}(x)\right] \;\geq\; \mathbf{E}\left[\mathrm{acc}(x) \cdot \left(\frac{\mathrm{acc}(x)}{q} - \frac{1}{h} \right) \right] \\
&= \frac{\mathbf{E}\left[\mathrm{acc}(x)^2\right]}{q} - \frac{\mathbf{E}\left[\mathrm{acc}(x)\right]}{h} \\
&\geq \frac{\mathbf{E}\left[\mathrm{acc}(x)\right]^2}{q} - \frac{\mathbf{E}\left[\mathrm{acc}(x)\right]}{h} \;=\; \mathrm{acc} \cdot \left(\frac{\mathrm{acc}}{q} - \frac{1}{h} \right)\ .
\end{aligned}$$

This establishes (1). Above, we used (3), Lemma 2 and also the fact that $\mathbf{E}\left[\mathrm{acc}(x)\right] = \mathrm{acc}$.

We proceed to the proof of (3). For any input x, with probabilities taken over the coin tosses of $\mathsf{F_A}$ we have

$$\begin{aligned} \mathrm{frk}(x) &= \Pr\left[\, I = I' \wedge I \geq 1 \wedge h_I \neq h'_I \,\right] \\ &\geq \Pr\left[\, I = I' \wedge I \geq 1 \,\right] - \Pr\left[\, I \geq 1 \wedge h_I = h'_I \,\right] \\ &= \Pr\left[\, I = I' \wedge I \geq 1 \,\right] - \frac{\Pr\left[\, I \geq 1 \,\right]}{h} \\ &= \Pr\left[\, I = I' \wedge I \geq 1 \,\right] - \frac{\mathrm{acc}(x)}{h} \,. \end{aligned}$$

It remains to show that $\Pr\left[\, I = I' \wedge I \geq 1 \,\right] \geq \mathrm{acc}(x)^2/q$. Let $\mathcal{R}$ denote the set from which A draws its coins at random. For each $i \in \{1, \ldots, q\}$ let $X_i \colon \mathcal{R} \times H^{i-1} \to [0,1]$ be defined by setting $X_i(\rho, h_1, \ldots, h_{i-1})$ to

$$\Pr\left[\; J = i \;:\; h_i, \ldots, h_q \xleftarrow{\$} H \;;\; (J, \sigma) \leftarrow \mathsf{A}(x, h_1, \ldots, h_q; \rho) \;\right]$$

for all $\rho \in \mathcal{R}$ and $h_1, \ldots, h_{i-1} \in H$. Regard X_i as a random variable over the uniform distribution on its domain. Then

$$\begin{aligned} &\Pr\left[\, I = I' \wedge I \geq 1 \,\right] \\ &\quad= \sum_{i=1}^{q} \Pr\left[\, I = i \wedge I' = i \,\right] \\ &\quad= \sum_{i=1}^{q} \Pr\left[\, I = i \,\right] \cdot \Pr\left[\, I' = i \,\middle|\, I = i \,\right] \\ &\quad= \sum_{i=1}^{q} \sum_{\rho, h_1, \ldots, h_{i-1}} X_i(\rho, h_1, \ldots, h_{i-1})^2 \cdot \frac{1}{|\mathcal{R}| \cdot |H|^{i-1}} \\ &\quad= \sum_{i=1}^{q} \mathbf{E}\left[X_i^2\right] \;\geq\; \sum_{i=1}^{q} \mathbf{E}\left[X_i\right]^2 \,, \end{aligned}$$

where in the last step we used Lemma 2. Now let $x_i = \mathbf{E}\left[X_i\right]$ for $i \in \{1, \ldots, q\}$, and apply Lemma 3. We get

$$\sum_{i=1}^{q} \mathbf{E}\left[X_i\right]^2 \;\geq\; \frac{1}{q} \cdot \left(\sum_{i=1}^{q} \mathbf{E}\left[X_i\right] \right)^2 \;=\; \frac{1}{q} \cdot \mathrm{acc}(x)^2 \,.$$

This completes the proof of (3) and thus of (1). We now show how to obtain (2). Using (1) we have

$$\left(\mathrm{acc} - \frac{q}{2h}\right)^2 \;=\; \mathrm{acc}^2 - \frac{q}{h} \cdot \mathrm{acc} + \frac{q^2}{4h^2} \;\leq\; q \cdot \mathrm{frk} + \frac{q^2}{4h^2} \,.$$

Taking the square root of both sides, and using the fact that $\sqrt{a+b} \leq \sqrt{a} + \sqrt{b}$ for any real numbers $a, b \geq 0$, we get

$$\mathrm{acc} - \frac{q}{2h} \;\leq\; \sqrt{q \cdot \mathrm{frk}} + \sqrt{\frac{q^2}{4h^2}} \;=\; \frac{q}{2h} + \sqrt{q \cdot \mathrm{frk}} \,.$$

Re-arranging terms yields (2). □

4. MULTI-SIGNATURES

Here we provide our definitions for multi-signatures with security in the plain public-key model.

The model. Consider a group of signers signing the same message m, each having as input its own public and secret key as well as a list of the public keys of the other signers. The signers want to interact in a protocol which eventually outputs a compact signature σ that represents the signature of each individual signer on the message m. We assume the signers are connected to each other via point-to-point links over which they can send messages. We do not assume these links are secure (that is, they are neither private nor authenticated) and we do not assume a broadcast primitive. The signers interact for some number of rounds. In each round, view a signer as receiving a (but not necessarily the same) message from every other signer, performing some computation, and then sending a message to every other signer, except that in the first round the "received message" is the party's input (and so is not really received) and in the last round the "sent message" is a local output (and so is not really sent). The local output is either $\perp$ to indicate failure or is the compact signature σ. Instances of the protocol may be executed concurrently, with one signer possibly participating in several concurrent instances at the same time.

In describing protocols, we will have each signer assign indices $1, \ldots, n$ to the signers, with itself being signer 1. We clarify that these are local references to the cosigners participating in this protocol instance. (That is, each signer in this protocol instance chooses its own indexing, so that signer 3 on my list and your list may not be the same. Think of the index a signer gives to its cosigners as locally identifying the link over which they communicate.) These indices have no certified relationship with the public keys. In particular, they are not identities.

Formally a multi-signature scheme $\mathsf{MS} = (\mathsf{Pg}, \mathsf{Kg}, \mathsf{Sign}, \mathsf{Vf})$ consists of four algorithms. A central authority runs the parameter generation algorithm Pg to generate the system-wide parameters *par*. Each signer independently generates its own public and private key pair via $(pk, sk) \xleftarrow{\$} \mathsf{Kg}(par)$. We stress that this is a non-interactive process that can be performed by any signer at any given time. New signers can join the system at will, and need not engage in expensive protocols with a CA or with other signers to prove knowledge of the corresponding secret key before participating in signing protocols. The Sign algorithm represents the signing protocol as indicated above. The verification algorithm Vf takes as input a multiset of public keys $L = \{pk_1, \ldots, pk_n\}$, the message m and a candidate signature σ, and outputs 1 if σ is a valid signature for L and m, or outputs 0 otherwise. (Because users may let their keys depend on those of other users, we explicitly allow them to be the same by modeling L as a multiset.) We have the obvious correctness requirement, namely that if a group of signers begin their interaction with public keys $L = \{pk_1, \ldots, pk_n\}$ and message m, and all signers follow the protocol (meaning, perform their computations according to Sign) then all have local output σ such that Vf returns 1 on input L, σ.

When describing protocols we will not specify the algorithms directly but instead describe them informally by saying what parties receive, compute and send in each round.

Security. The notion of security requires that it be infeasible to forge multi-signatures involving at least one honest signer. As in previous works [31, 12, 29] we can in fact assume there is a single honest signer. Our adversary will be viewed as having effectively corrupted all other signers. It can choose their public keys as it likes, even as a function of the public key of the honest signer, and can interact arbitrarily with the honest signer in any number of concurrent instances before outputting its forgery attempt. In somewhat more detail, we consider the following three-phase game associated to multi-signature scheme $\mathsf{MS} = (\mathsf{Pg}, \mathsf{Kg}, \mathsf{Sign}, \mathsf{Vf})$ and adversary (forger) F:

Setup: The game chooses system-wide parameters $par \xleftarrow{\$} \mathsf{Pg}$ and a key-pair $(pk^*, sk^*) \xleftarrow{\$} \mathsf{Kg}(par)$ for the for the "target" (honest) signer. The target public key pk^* is given as input to the forger F.

Attack: F can start a protocol instance with the honest signer by providing the latter with a message m and a multiset $L = \{pk_1, \ldots, pk_n\}$ of purported cosigners, where pk^* occurs in L at least once. It can choose these public keys as it wishes, including as a function of pk^* and previous protocol flows. In interacting with the honest signer, F will play the role of all signers in L other than one instance of pk^*, sending messages to, and receiving messages from, the honest signer. The forger can schedule an arbitrary number of protocol instances concurrently, interacting with "clones" of the honest signer, where each clone maintains its own state and uses its own coins but all use the keys pk^*, sk^* and follow the protocol (meaning use algorithm Sign) to compute their responses to received messages. When the honest signer terminates then its local output (whether $\perp$ or a compact signature) is returned to F.

Forgery: At the end of its execution, F outputs a multiset $L = \{pk_1, \ldots, pk_n\}$, a message m and a forged signature σ. The forger is said to win the game if $\mathsf{Vf}(L, m, \sigma) = 1$, $pk^* \in L$ and F never initiated a signing protocol with L, m.

The advantage of algorithm F in breaking MS, denoted as $\mathbf{Adv}^{\text{uf-cma}}_{\mathsf{MS}}(\mathsf{F})$, is defined as the probability that F wins the above game, where the probability is taken over the coin tosses of the forger, the honest signer, and the setup phase. We say that a forger F $(t, q_{\mathrm{S}}, N, \epsilon)$-breaks MS if F runs in time at most t, F initiates at most q_{S} signing protocols with the honest signer, the number of public keys in the multiset L involved in any signing query or in the forgery is at most N, and $\mathbf{Adv}^{\text{uf-cma}}_{\mathsf{MS}}(\mathsf{F}) \geq \epsilon$. The scheme MS is said to be $(t, q_{\mathrm{S}}, N, \epsilon)$-secure if no forger $(t, q_{\mathrm{S}}, N, \epsilon)$-breaks it. In the random oracle model, the Sign and Vf algorithms, as well as the adversary, additionally have access to a random oracle $\mathrm{H} : \{0,1\}^* \to D$, where D is a set possibly depending on the system parameters. The additional parameter q_{H} denotes the maximum number of F's random oracle queries. (If there is more than one random oracle, we mean the total number of queries to all random oracles.) We say that F $(t, q_{\mathrm{S}}, q_{\mathrm{H}}, N, \epsilon)$-breaks MS in the random oracle model, and that MS is $(t, q_{\mathrm{S}}, q_{\mathrm{H}}, N, \epsilon)$-secure in the random oracle model.

Discussion. Note that the game described above does not require the adversary to fix the public keys of all signers in the system at the beginning of the game (as is required by the notions of [31, 11]), or to submit the secret keys of all corrupt signers to a special certification oracle (as is required by the notions of [11, 29]). Rather, our model allows the adversary to dynamically add new signers to the system, using arbitrary public keys that may depend on the target public key or on previous signing interactions. It thereby avoids the KOSK assumption and does not presume expensive proof of knowledge protocols to be performed with the CA. This security notion reflects a real-world system with the desirable features that new signers can join on-the-fly using self-generated keys, and that existing (external) CA infrastructure can be reused for the certification of these keys.

Interactive aggregate signatures. Although multi-signatures are presented as being about a bunch of signers signing a single common message, one can consider something more general where each party has its own message. Let us call this an *interactive aggregate signature* (IAS). Here party i has message M_i. The parties start knowing all messages and each other's public keys, interact, and finally produce an aggregate signature that is supposed to validate that party i signed M_i for all i involved. An IAS scheme can be viewed either as a generalization of a sequential aggregate signature scheme [30] where the interaction between signers is arbitrary rather than sequential, or as an extension of a MS scheme where each signer has a different message. IAS schemes potentially have more applications than MS schemes. However, we observe that IAS and MS are equivalent in the sense that any scheme for one is easily turned into a scheme for the other. Indeed, we can implement IAS given a MS scheme by using as message in the latter the tuple of all messages and public keys of the former. On the other hand, obviously, we can implement an MS scheme given an IAS scheme by setting the messages of all parties to the single message of the MS scheme. For this reason we do not explicitly consider IAS schemes further, but it is worthwhile to note that that the single-message restriction of an MS scheme is not really limiting. We also think IAS schemes are interesting in that they unify aggregate and multi-signatures, both of which are special cases of IASs.

5. OUR MULTI-SIGNATURE SCHEME

Our scheme is based on Schnorr's signature scheme [42]. Let $\mathbb{G}$ be a cyclic group of prime order p, and let g be a generator of $\mathbb{G}$. Recall that a Schnorr signature of a message m under public key $X \in \mathbb{G}$ is a pair $(R, s) \in \mathbb{G} \times \mathbb{Z}_p$ such that $g^s = RX^c$ in $\mathbb{G}$, where c is the response to a random oracle query on $R \| m$. A first idea to aggregate signatures may be to let a signature under keys $L = \{X_1, \ldots, X_n\}$ of message m be a pair (R, s) such that $g^s = R \prod_{i=1}^{n} X_i^c$, where $c = \mathrm{H}(R \| \langle L \rangle \| m)$ is determined by a random oracle. Without restrictions on key generation however, this type of scheme is vulnerable to a well-known attack [24, 27, 32, 31] where a corrupt signer chooses $x_1 \xleftarrow{\$} \mathbb{Z}_p$ and sets its public key $X_1 \leftarrow g^{x_1} \cdot \prod_{i=2}^{n} X_i^{-1}$. This way, x_1 essentially becomes the "secret key" for the entire group of signers $L = \{X_1, \ldots, X_n\}$: signer 1 can by himself sign any message m in name of the entire group L by choosing $r \xleftarrow{\$} \mathbb{Z}_p$ and computing $(R = g^r\ ,\ s = cx_1 + r \bmod p)$ where $c = \mathrm{H}(R \| \langle L \rangle \| m)$.

We counteract this attack by using a different value c_i in the exponent of each public key X_i, so that the verification equation becomes $g^s = R \prod_{i=1}^{n} X_i^{c_i}$, where the values for c_i are determined by independent random oracle queries $c_i = \mathrm{H}(X_i \| R \| \langle L \rangle \| m)$. With the help of our general Forking Lemma (see Lemma 1), we succeed in extracting from any forger the discrete logarithm of the target public key. The way we apply the Forking Lemma is particularly interesting because we need certain random oracle responses to be the same in both executions of the adversary, even though the corresponding queries may not occur until *after* the fork.

A second problem is that, in order to respond to the forger's signature queries, the simulator needs to know the value of R before the forger does so that it can program the random oracle. The value of R is typically computed as the product of individual shares of R chosen by each signer. Unless the target signer is the last to reveal his share (which we cannot assume to always be the case), the forger knows R before the simulator does, enabling the forger to perform a random oracle query involving R and thereby to prevent the simulator from programming this entry later on. We overcome this problem by letting signers first "commit" to their share of R through an additional random oracle query. The simulator, who sees all random oracle queries, can therefore look up the individual shares of R before the forger can, and can thus correctly program the random oracle.

We now proceed to describe the scheme and analyze its security. We begin by recalling some necessary definitions.

THE DISCRETE LOGARITHM ASSUMPTION. Let $\mathbb{G}$ be a multiplicative group of prime order p, and let $\mathbb{G}^* = \mathbb{G} \setminus \{1\}$. The advantage of algorithm A in solving the discrete logarithm problem in $\mathbb{G}$ is defined as

$$\mathbf{Adv}^{\mathrm{dlog}}_{\mathbb{G}}(\mathsf{A}) = \Pr\left[\, g^x = y \mid g \xleftarrow{\$} \mathbb{G}^* \,;\, y \xleftarrow{\$} \mathbb{G} \,;\, x \xleftarrow{\$} \mathsf{A}(y) \,\right] \;.$$

We say that A (t, ϵ)-solves the discrete logarithm problem in $\mathbb{G}$ if it runs in time at most t and $\mathbf{Adv}^{\mathrm{dlog}}_{\mathbb{G}}(\mathsf{A}) \geq \epsilon$, and we say that the discrete logarithm problem in $\mathbb{G}$ is (t, ϵ)-hard if no algorithm A (t, ϵ)-solves it.

THE SCHEME. Let $k = \lfloor \log_2 p \rfloor$, let $l_0, l_1 \in \mathbb{N}$, and let $\mathrm{H}_0 : \{0,1\}^* \to \{0,1\}^{l_0}$ and $\mathrm{H}_1 : \{0,1\}^* \to \{0,1\}^{l_1}$ be random oracles. To these, we associate the multi-signature scheme $\mathsf{MS\text{-}BN} = (\mathsf{Pg}, \mathsf{Kg}, \mathsf{Sign}, \mathsf{Vf})$ as follows:

Parameter generation. A trusted center chooses a random generator $g \xleftarrow{\$} \mathbb{G}^*$ and publishes $(\mathbb{G}, p, g)$ as system-wide parameters.

Key generation. Each signer runs the Kg algorithm to generate a random secret key $x \xleftarrow{\$} \mathbb{Z}_p$ and the corresponding public key $X \leftarrow g^x$.

Signing. Let X_1 and x_1 be the public and private key of a signer, let m be the message to be signed, and let $X_2, \ldots, X_n$ be the public keys of all other cosigners. We recall that the indices $1, \ldots, n$ are merely local references to cosigners, defined by one signer within one protocol instance. These indices are not tied to public keys in a global way, and in particular are not unique identities of signers. The communication proceeds in a number of rounds, where in each round each signer receives a message from every other signer, performs some local computation, and sends a message to every other signer.

Round 1:

- Local input: x_1, $L = \{X_1, \ldots, X_n\}$, m
- Computation: Choose $r_1 \xleftarrow{\$} \mathbb{Z}_p$, compute $R_1 \leftarrow g^{r_1}$ and query $t_1 \leftarrow \mathrm{H}_0(R_1)$.
- Send to signer i: t_1

Round 2:

- Receive from signer i: t_i
- Send to signer i: R_1

Round 3:

- Receive from signer i: R_i
- Computation: For all $2 \leq i \leq n$, check that $t_i = \mathrm{H}_0(R_i)$. Abort the protocol with local output $\perp$ if this is not the case; otherwise, compute $R \leftarrow \prod_{i=1}^{n} R_i$, query $c_1 \leftarrow \mathrm{H}_1(X_1 \| R \| \langle L \rangle \| m)$ where $\langle L \rangle$ is a unique encoding of L (e.g. the sequence of keys in lexicographic order), and compute $s_1 \leftarrow x_1 c_1 + r_1 \bmod p$.
- Send to signer i: s_1

Round 4:

- Receive from signer i: s_i
- Computation: $s \leftarrow \sum_{i=1}^{n} s_i \bmod p$
- Local output: the signature $\sigma = (R, s)$

Verification. Given a multiset of public keys $L = \{X_1, \ldots, X_n\}$, message m and signature $\sigma = (R, s)$, the verifier computes $c_i \leftarrow \mathrm{H}_1(X_i \| R \| \langle L \rangle \| m)$ for all $1 \leq i \leq n$. He accepts the signature if $g^s = R \prod_{i=1}^{n} X_i^{c_i}$, and rejects otherwise.

EFFICIENCY. An overview comparing the efficiency of our scheme to that of other (provably secure) multi-signature schemes is given in Table 1. Compared to the $\mathsf{MS\text{-}MOR}$ scheme [31], our $\mathsf{MS\text{-}BN}$ scheme avoids the expensive interactive key generation protocol, while offering considerably shorter public keys and maintaining the same signature length and signing/verification costs. Moreover, our scheme allows for concurrent signing sessions at the cost of one extra round of interaction (and the computation of some hash values). Compared to the $\mathsf{MS\text{-}Bo}$ scheme [11], our scheme has about twice the signature size, yet offers faster verification. Our scheme beats the $\mathsf{MS\text{-}LOSSW}$ scheme [29] in all costs. Moreover, our scheme has the advantage over $\mathsf{MS\text{-}Bo}$ and $\mathsf{MS\text{-}LOSSW}$ of not relying on pairings to be defined over the underlying group. On the other hand, the signing protocol in our scheme has more rounds of interaction than in the other schemes.

We note that when we motivated communication reduction in the Introduction (in particular for mobile devices), we were referring to the size of the signature, not to the communication cost of the protocol that computes the signature. This is because it is the signatures that are frequently communicated. In fact the protocol may take place over a a different, cheaper medium, yet produce signatures (e.g. certificates) that are placed on mobile devices and then frequently transmitted by these devices.

SECURITY. The following theorem implies that the $\mathsf{MS\text{-}BN}$ scheme meets our definition of security in the plain public-key model (i.e., without the KOSK assumption).

THEOREM 4. *If there exists a $(t, q_{\mathrm{S}}, q_{\mathrm{H}}, N, \epsilon)$-forger F in the random-oracle model against the MS scheme $\mathsf{MS\text{-}BN}$ described above, then there exists an algorithm B that (t', ϵ')-breaks the discrete logarithm problem in $\mathbb{G}$, where*

$$\begin{aligned} \epsilon' &\geq \frac{\epsilon^2}{q_{\mathrm{H}} + q_{\mathrm{S}}} - \frac{2q_{\mathrm{H}} + 16N^2 q_{\mathrm{S}}}{2^{l_0}} - \frac{8Nq_{\mathrm{S}}}{2^k} - \frac{1}{2^{l_1}} \;, \quad (4) \\ t' &= 2t + q_{\mathrm{S}} t_{\exp} + O((q_{\mathrm{S}} + q_{\mathrm{H}})(1 + q_{\mathrm{H}} + Nq_{\mathrm{S}})) \end{aligned}$$

and $t_{\exp}$ is the time of an exponentiation in $\mathbb{G}$. ▮

PROOF OF THEOREM 4. The idea of the proof is to use our Forking Lemma to obtain from the forger F two forgeries (R, s) and (R', s') satisfying

$$g^s \;=\; R \prod_{i=1}^{n} X_i^{c_i} \quad \text{and} \quad g^{s'} \;=\; R \prod_{i=1}^{n} X_i^{c_i'} \;,$$

such that $c_i = c_i'$ if X_i is the target public key X^*, and $c_i \neq c_i'$ for all other keys. We can then extract the discrete logarithm of X^* by dividing the two equations above. Special care must be taken however in responding F's random oracle queries so that the above relations between c_i and c_i' are ensured. In particular, F may not ask the queries defining c_i and c_i' until *after* the fork, where the two executions of F have already diverged. We overcome this by fixing the response values to these queries *before* the fork, and by recognizing the queries when they actually occur after the fork. It is mainly due to the modularity of our simplified Forking Lemma that the complexity of the proof is kept manageable.

We are now ready to present the actual proof. Given a $(t, q_S, q_H, N, \epsilon)$-forger F, consider the following algorithm A. On inputs $g \in \mathbb{G}^*$, $X^* \in \mathbb{G}$ and $h_1, \ldots, h_{q_H+q_S} \in \{0,1\}^{l_1}$, algorithm A runs the forger F on input system parameters $par = (\mathbb{G}, p, g)$ and target public key $pk^* = X^*$. Algorithm A initializes counters ctr_1, ctr_2 to zero, and maintains initially empty associative arrays $T_0[\cdot], T_1[\cdot,\cdot], T_2[\cdot]$. It assigns $T_2[X^*] \leftarrow 0$ and answers F's oracle queries as follows. Tables T_0 and T_1 are used to simulate random oracles H_0 and H_1, respectively, while T_2 assigns a unique index $1 \le i \le q_H + Nq_S$ to each public key X occurring either as a cosigner's public key in one of F's signature queries, or as the first item in the argument of one of F's queries to H_1. Algorithm A assigns index 0 to the target public key X^* by setting $T_2[X^*] \leftarrow 0$. It responds to F's oracle queries as follows:

- $\mathrm{H}_0(R)$: If $T_0[R]$ is undefined, then A chooses $T_0[R] \xleftarrow{\$} \{0,1\}^{l_0}$. It returns $T_0[R]$ to F.

- $\mathrm{H}_1(X\|Q)$: If $T_2[X]$ is undefined then A increases ctr_2 and sets $T_2[X] \leftarrow ctr_2$. Let $i = T_2[X]$. If $T_1[i, Q]$ has not yet been defined, then A immediately assigns random values to *all* entries $T_1[j, Q]$ for $1 \le j \le q_H + Nq_S$, increases ctr_1 and assigns $T_0[0, Q] \leftarrow h_{ctr_1}$. (If the argument of the query cannot be parsed as $X\|Q$, then A simply returns a random element of $\{0,1\}^{l_1}$, preserving consistency if the same query is asked again.)

- Signing query with public keys L and message m: If $X^* \notin L$ then algorithm A returns $\perp$ to F; otherwise, it parses the elements of L as $\{X_1 = X^*, X_2, \ldots, X_n\}$ and continues as follows. First, it checks for all $2 \le i \le n$ whether $T_2[X_i]$ has already been defined; it increases ctr_2 and sets $T_2[X_i] \leftarrow ctr_2$ if not. Then, A increases counter ctr_1 and sets $c_1 \leftarrow h_{ctr_1}$. It chooses $s_1 \xleftarrow{\$} \mathbb{Z}_p$, computes $R_1 \leftarrow g^{s_1} X_1^{-c_1}$ and sends $t_1 = \mathrm{H}_0(R_1)$ to all cosigners.

  After receiving $t_2, \ldots, t_n$ from F (who's playing the role of the cosigners), A searches in table T_0 for the values R_i so that $t_i = T_0[R_i]$. If no such R_i can be found for some $2 \le i \le n$, then A sets a flag $alert \leftarrow \mathtt{true}$ and sends R_1 to all cosigners. If more than one such value is found for some R_i, then it sets $bad_1 \leftarrow \mathtt{true}$, aborts the execution of F and halts with output $(0, \varepsilon)$. Otherwise, A computes $R \leftarrow \prod_{i=1}^n R_i$ and checks whether $T_1[0, Q]$ has already been defined for $Q = R\|\langle L\rangle\|m$. If so, it sets $bad_2 \leftarrow \mathtt{true}$, aborts the execution of F and halts with output $(0, \varepsilon)$. If not, it sets $T_1[0, Q] \leftarrow c_1$, chooses $T_1[i, Q] \xleftarrow{\$} \{0,1\}^{l_1}$ for all $1 \le i \le q_H + Nq_S$, and sends R_1 to all cosigners.

  If, after receiving $R_2, \ldots, R_n$, there exists an index $1 \le i \le n$ such that $\mathrm{H}_0(R_i) \ne t_i$, then A stops the signing protocol returning $\perp$. If $alert = \mathtt{true}$ while $\mathrm{H}_0(R_i) = t_i$ for all $1 \le i \le n$, then it sets $bad_3 \leftarrow \mathtt{true}$, aborts the execution of F and halts outputting $(0, \varepsilon)$. Otherwise, it sends s_1 to all cosigners.

  After receiving $s_2, \ldots, s_n$, A computes $s \leftarrow \sum_{i=1}^n s_i \bmod p$ and returns (R, s) as the signature.

Eventually, F outputs a forged signature (R, s) together with multiset $L = \{X_1, \ldots, X_n\}$ and message m. Algorithm A first performs additional queries $\mathrm{H}_1(X_i\|R\|\langle L\rangle\|m)$ for $1 \le i \le n$, thereby making sure that $T_2[X_i]$ is defined. Let $1 \le J \le q_H + q_S$ be the index such that $T_1[0, R\|\langle L\rangle\|m] = h_J$, and let n^* be the number of times that X^* occurs in L. If F's forgery is valid, algorithm A halts returning $(J, (R, h_J, s, n^*))$; if not, it halts returning $(0, \varepsilon)$.

Let $\Pr[\, bad_i \,]$ denote the probability of the event that flag bad_i gets set to $\mathtt{true}$. Consider set $H = \{0,1\}^{l_1}$ and input generator IG that returns random elements $g, X^* \xleftarrow{\$} \mathbb{G}$. We bound the accepting probability acc of A with respect to these, as defined in Lemma 1, as follows:

$$\begin{aligned}
\mathrm{acc} &\ge \epsilon - \Pr[\, bad_1 \,] - \Pr[\, bad_2 \,] - \Pr[\, bad_3 \,] \\
&\ge \epsilon - \frac{(q_H + Nq_S + 1)^2}{2^{l_0+1}} \\
&\quad - \sum_{i=1}^{q_S} \left(\frac{q_H + Nq_S}{2^k} + \frac{q_H + q_S}{2^k} + \frac{N}{2^{l_0}} \right) \\
&\ge \epsilon - \frac{(q_H + Nq_S + 1)^2}{2^{l_0}} - \frac{2q_S(q_H + Nq_S)}{2^k} .
\end{aligned}$$

We clarify how the bounds in the second inequality were obtained. If at some point in the execution of F two values $R_i \ne R_i'$ are found such that $t_i = \mathrm{H}_0(R_i) = \mathrm{H}_0(R_i')$, then clearly at least one collision must have occurred in H_0. However, all response values of H_0 are chosen uniformly at random from $\{0,1\}^{l_0}$, and since there are at most $q_H + Nq_S$ queries to H_0, the probability that at least one collision occurs is at most $((q_H + Nq_S)(q_H + Nq_S + 1)/2)/2^{l_0} \le (q_H + Nq_S + 1)^2/2^{l_0}$.

To cause bad_2 to be set to $\mathtt{true}$ during the i-th signing query, we distinguish between the case that $\mathrm{H}_0(R_1)$ was previously queried by the forger, and the case that it wasn't. In the first case, F probably knows R and may have deliberately queried $\mathrm{H}_1(X\|R\|\langle L\rangle\|m)$ for some X. But since R_1 was chosen by A independently from F's view at the beginning of the signing protocol, the probability that F queried $\mathrm{H}_0(R_1)$ is at most $(q_H + Nq_S)/p \le (q_H + Nq_S)/2^k$. In the second case, F's view is completely independent of R_1, and hence of R. The probability that R occurred by chance in a previous query to H_1 or was set by A in one of the $i-1$ previous signature simulations is at most $(q_H + q_S)/p \le (q_H + q_S)/2^k$.

Lastly, in order to set $bad_3 = \mathtt{true}$, F must have predicted the value of $\mathrm{H}_0(R_i)$ for at least one $1 \le i \le n$, which it can do with probability at most $N/2^{l_0}$. The third inequality follows from simple rearranging of terms after assuming (without loss of generality) that $q_H, q_S, N > 0$.

Now consider an algorithm B that on input X^* runs the forking algorithm $\mathsf{F}_\mathsf{A}(X^*)$, which with probability frk returns $(1, (R, h, s, n^*), (R', h', s', n'^*))$ with $h \ne h'$. These forgeries are such that

$$g^s = R \prod_{i=1}^{n} X_i^{c_i} \quad \text{and} \quad g^{s'} = R' \prod_{i=1}^{n'} X_i'^{c_i'}$$

where $L = \{X_1, \ldots, X_n\}$ and m are the public keys and the message involved in F's forgery and $c_i = \mathrm{H}_1(X_i\|R\|\langle L\rangle\|m)$ are the relevant random oracle responses from the first run. Let $I^* \subseteq \{1, \ldots, n\}$ be the set of indices such that $X_i = X^*$. Variables $L', X_1', \ldots, X_{n'}', m', c_1', \ldots, c_{n'}', I'^*$ are defined analogously for the second run of F. We will show later that, due to the way that A simulates F's environment, it must hold that $n = n'$, that $L = L'$, that $I^* = I'^*$, that $n^* = n'^*$, that $c_i = c'_i$ for $i \notin I^*$, and that $c_i = h$ and $c'_i = h'$ for $i \in I^*$. Dividing the two equations above then gives

$$g^{s-s'} = \prod_{i \in I^*} (X^*)^{h-h'} = (X^*)^{n^*(h-h')} \; ,$$

so that B can compute the discrete logarithm of X^* as $(s - s')/(n^*(h - h')) \bmod p$. The probability that algorithm B succeeds in doing so is given by

$$\begin{aligned}
\epsilon' &\geq \mathrm{frk} \\
&\geq \frac{\mathrm{acc}^2}{q_{\mathrm{H}} + q_{\mathrm{S}}} - \frac{1}{2^{l_1}} \\
&\geq \frac{\epsilon^2}{q_{\mathrm{H}} + q_{\mathrm{S}}} - \frac{2(q_{\mathrm{H}} + Nq_{\mathrm{S}} + 1)^2}{(q_{\mathrm{H}} + q_{\mathrm{S}}) \cdot 2^{l_0}} - \frac{4q_{\mathrm{S}}(q_{\mathrm{H}} + Nq_{\mathrm{S}})}{(q_{\mathrm{H}} + q_{\mathrm{S}}) \cdot 2^k} - \frac{1}{2^{l_1}} \\
&\geq \frac{\epsilon^2}{q_{\mathrm{H}} + q_{\mathrm{S}}} - \frac{2q_{\mathrm{H}} + 16N^2 q_{\mathrm{S}}}{2^{l_0}} - \frac{8Nq_{\mathrm{S}}}{2^k} - \frac{1}{2^{l_1}} \; ,
\end{aligned}$$

where again we assume without loss of generality that $q_{\mathrm{H}}, q_{\mathrm{S}}, N > 0$. The theorem follows.

We still have to argue why the equalities between all the variables in both runs of A hold. In the case that F returned $(1, (R, h, s, n^*), (R, h', s', n'^*))$, let J be the index that A returned after both executions by $\mathsf{F_A}$. In A's first execution, $h_J = h$ is assigned to $T_1[0, R \| \langle L \rangle \| m] = \mathrm{H}_1(X^* \| R \| \langle L \rangle \| m)$ at the moment when F makes its first query $\mathrm{H}_1(X \| R \| \langle L \rangle \| m)$ for *some* public key X (so not necessarily X^*), where $L = \{X_1, \ldots, X_n\}$. Likewise, in the second run, $h'_J = h'$ is assigned to $T_1[0, R' \| \langle L' \rangle \| m']$ when F queries $\mathrm{H}_1(X' \| R' \| \langle L' \rangle \| m')$ for some public key X', where $L' = \{X'_1, \ldots, X'\}$. Up to the point of this hash query, however, the environments of F provided by A in the first and the second run are identical, because A uses the same inputs, random tape and values $h_1, \ldots, h_{J-1}$ to generate F's inputs, random tape and oracle responses. Therefore, the two executions of F are identical up to this point, and in particular the arguments of both hash queries must be the same, implying that $R = R'$, $L = L'$, $n = n'$, $X_i = X'_i$ and $m_i = m'_i$ for $1 \leq i \leq n$. Moreover, the entries for $X_1, \ldots, X_n$ in T_2 are assigned *at the latest* when parsing the arguments of this hash query, causing the values of $T_2[X_i]$ to be the same in both runs as well. The forger's other queries $\mathrm{H}_1(X_i \| R \| \langle L \rangle \| m)$ may not occur until much later, but the response values $T_1[T_2[X_i], R \| \langle L \rangle \| m]$ for these queries are chosen *before* the fork, and hence are the same in both runs as well. Therefore, it holds that $c_i = c'_i$ for all $1 \leq i \leq n$, that $c_i = h_J = h$ for $i \in I$, and that $c'_i = h'_J = h'$ for $i \in I$, which concludes the proof.

The running time t' of B is twice that of A plus the time needed to compute the discrete logarithm $(s - s')/(n^*(h - h')) \bmod p$. The running time of A is the running time t of F plus the time needed to answer $q_{\mathrm{H}} + Nq_{\mathrm{S}}$ random oracle queries and q_{S} signature queries. We assume that exponentiations in $\mathbb{G}$ take time $t_{\exp}$, and all other operations take unit time. Each random oracle query may cause A to perform $O(1 + q_{\mathrm{H}} + Nq_{\mathrm{S}})$ unit-time operations. Each signature query involves one multi-exponentiation in $\mathbb{G}$ and $O(1 + q_{\mathrm{H}} + Nq_{\mathrm{S}})$ unit-time operations. Therefore, we have $t' = 2t + q_{\mathrm{S}} t_{\exp} + O((q_{\mathrm{S}} + q_{\mathrm{H}})(1 + q_{\mathrm{H}} + Nq_{\mathrm{S}}))$. $\square$

Reduction tightness. The reduction presented above is tighter than that of the MS-MOR scheme [31], but is still not tight, as can be seen from (4). In comparison, the security proof of the MS-MOR scheme requires two applications of the forking technique (once to extract the secret keys of corrupt players, and once to obtain two forgeries) and $q_{\mathrm{H}} \cdot q_{\mathrm{S}}$ rewindings (to simulate signing protocols) of the forger, yielding a considerable loss in the tightness of the security reduction. The pairing-based MS-Bo and MS-LOSSW schemes do not have tight security reductions either.

6. FURTHER RESULTS

Our scheme is based on the Schnorr signature scheme [42], but our techniques can be applied to other schemes following the Fiat-Shamir paradigm as well. However, unlike the case of standard signatures [1], a three-move identification scheme does not automatically give rise to a multi-signature scheme: compression of signatures requires a special homomorphism to exist on conversation transcripts. A number of three-move identification schemes in the literature turn out to have such a homomorphism though. In particular, one can obtain efficient multi-signature schemes based on discrete logarithms from [36], based on RSA from [19], based on factoring from [16, 15, 33, 37], and based on pairings from [23]. In the full version of this paper [7], we also adapt ideas from Katz and Wang [26] to construct a scheme with a tight security reduction to the decisional Diffie-Hellman problem, at the cost of a slight increase in signing and verification time.

Acknowledgments

Mihir Bellare was supported by NSF grant CNS-0524765 and a gift from Intel Corporation. Gregory Neven is a Postdoctoral Fellow of the Research Foundation - Flanders (FWO-Vlaanderen), and was supported in part by the Concerted Research Action (GOA) Ambiorics 2005/11 of the Flemish Government and in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT.

7. REFERENCES

[1] M. Abdalla, J. H. An, M. Bellare, and C. Namprempre. From identification to signatures via the Fiat-Shamir transform: Minimizing assumptions for security and forward-security. *EUROCRYPT 2002*, LNCS 2332, Springer-Verlag.

[2] M. Abdalla and L. Reyzin. A new forward-secure digital signature scheme. *ASIACRYPT 2000*, LNCS 1976, Springer-Verlag.

[3] C. Adams, S. Farrell, T. Kause, and T. Monen. Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP). Internet Engineering Task Force RFC 4210, 2005.

[4] K. Barr and K. Asanovic. Energy aware lossless data compression. *MobiSys 2003*, ACM Press.

[5] M. Bellare and O. Goldreich. On defining proofs of knowledge. *CRYPTO 1992*, LNCS 740, Springer-Verlag.

[6] M. Bellare, C. Namprempre, and G. Neven. Unrestricted aggregate signatures. *Cryptology ePrint Archive*, Report 2006/285, 2006.

[7] M. Bellare and G. Neven. New multi-signatures and a general forking lemma. Full version of this paper, available from `http://www.cs.ucsd.edu/users/mihir`, 2006.

[8] M. Bellare and A. Palacio. GQ and Schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks. *CRYPTO 2002*, LNCS 2442, Springer-Verlag.

[9] M. Bellare, T. Ristenpart, and S. Yilek. Work in progress, 2006.

[10] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. *ACM CCS 1993*, ACM Press.

[11] A. Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. *PKC 2003*, LNCS 2567, Springer-Verlag.

[12] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. *EUROCRYPT 2003*, LNCS 2656, Springer-Verlag.

[13] D. Boneh, H. Shacham, and B. Lynn. Short signatures from the Weil pairing. *ASIACRYPT 2001*, LNCS 2248, Springer-Verlag.

[14] A. De Santis and G. Persiano. Zero-knowledge proofs of knowledge without interaction. *FOCS 1992*, IEEE Computer Society Press.

[15] U. Feige, A. Fiat, and A. Shamir. Zero knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, 1988.

[16] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. *CRYPTO 1986*, LNCS 263, Springer-Verlag.

[17] M. Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. *CRYPTO 2005*, LNCS 3621, Springer-Verlag.

[18] S. D. Galbraith, K. G. Paterson, and N. P. Smart. Pairings for cryptographers. *Cryptology ePrint Archive*, Report 2006/165, 2006.

[19] L. C. Guillou and J.-J. Quisquater. A "paradoxical" indentity-based signature scheme resulting from zero-knowledge. *CRYPTO 1988*, LNCS 403, Springer-Verlag.

[20] L. Harn. Group-oriented (t, n) threshold digital signature scheme and digital multisignature. *IEE Proceedings – Computers and Digital Techniques*, 141(5):307–313, 1994.

[21] R. Hayashi, T. Okamoto, and K. Tanaka. An RSA family of trap-door permutations with a common domain and its applications. *PKC 2004*, LNCS 2947, Springer-Verlag.

[22] J. Herranz and G. Sáez. Forking lemmas for ring signature schemes. *INDOCRYPT 2003*, LNCS 2947, Springer-Verlag.

[23] F. Hess. Efficient identity based signature schemes based on pairings. *SAC 2002*, LNCS 2595, Springer-Verlag.

[24] P. Horster, M. Michels, and H. Petersen. Meta-multisignatures schemes based on the discrete logarithm problem. *IFIP/SEC 1995*. Chapman & Hall.

[25] K. Itakura and K. Nakamura. A public-key cryptosystem suitable for digital multisignatures. *NEC Research & Development*, 71:1–8, 1983.

[26] J. Katz and N. Wáng. Efficiency improvements for signature schemes with tight security reductions. *ACM CCS 2003*, ACM Press.

[27] S. K. Langford. Weakness in some threshold cryptosystems. *CRYPTO 1996*, LNCS 1109, Springer-Verlag.

[28] C.-M. Li, T. Hwang, and N.-Y. Lee. Threshold-multisignature schemes where suspected forgery implies traceability of adversarial shareholders. *EUROCRYPT 1994*, LNCS 950, Springer-Verlag.

[29] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters. Sequential aggregate signatures and multisignatures without random oracles. *EUROCRYPT 2006*, LNCS 4004, Springer-Verlag.

[30] A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham. Sequential aggregate signatures from trapdoor permutations. *EUROCRYPT 2004*, LNCS 3027, Springer-Verlag.

[31] S. Micali, K. Ohta, and L. Reyzin. Accountable-subgroup multisignatures. *ACM CCS 2001*, ACM Press.

[32] M. Michels and P. Horster. On the risk of disruption in several multiparty signature schemes. *ASIACRYPT 1996*, LNCS 1163, Springer-Verlag.

[33] K. Ohta and T. Okamoto. A modification of the Fiat-Shamir scheme. *CRYPTO 1988*, LNCS 403, Springer-Verlag.

[34] K. Ohta and T. Okamoto. A digital multisignature scheme based on the Fiat-Shamir scheme. *ASIACRYPT 1991*, LNCS 739, Springer-Verlag.

[35] K. Ohta and T. Okamoto. Multi-signature schemes secure against active insider attacks. *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences*, E82-A(1):21–31, 1999.

[36] T. Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. *CRYPTO 1992*, LNCS 1751, Springer-Verlag.

[37] H. Ong and C.-P. Schnorr. Fast signature generation with a Fiat Shamir–like scheme. *EUROCRYPT 1990*, LNCS 473, Springer-Verlag.

[38] PKCS #10: Certification request syntax standard. RSA Data Security, Inc., 2000.

[39] D. Pointcheval, E. Brickell, S. Vaudenay, and M. Yung. Design validations for discrete logarithm based signature schemes. *PKC 2000*, LNCS 1751, Springer-Verlag.

[40] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000.

[41] J. Schaad. *Internet X.509 Public Key Infrastructure Certificate Request Message Format*, Internet Engineering Task Force RFC 4211, 2005.

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