Improving The Exact Security Of Digital Signature Schemes

Silvio Micali Leonid Reyzin [*]

August 15, 2000

Abstract

We put forward a new method of constructing Fiat-Shamir-like signature schemes that yields better "exact security" than the original Fiat-Shamir method. (We also point out, however, that such tight security does not make our modified schemes always preferable to the original ones. Indeed, there exist particularly efficient Fiat-Shamir-like schemes that, though only enjoying "loose security," by using longer keys may *provably* provide more security at a lower computational cost than their "tight-security" counterparts.)

Keywords: digital signatures, ID schemes, random oracles, exact security

Note: A preliminary version of this paper appears in [MR99]. A version that is similar to this one, but omits some proofs, will appear in the Journal of Cryptology. This is the full version.

1 Introduction

1.1 Exact Security of Signature Schemes

Goldwasser, Micali and Rivest's ([GMR88]) classical notion of security for a digital signature scheme is asymptotic in nature. In essence, a proof of security amounts to a reduction from forging a signature to solving a computationally hard problem: if a polynomial-time forger exists, then we can use it to solve the hard problem in polynomial time.

It has been often pointed out that this asymptotic approach, which uses notions such as "polynomial time" and "sufficiently large," is too coarse for practical security recommendations. Knowing that no polynomial-time adversary has a better than exponentially small chance of forgery for a sufficiently large security parameter does not provide one with an answer to the practical problem of finding the appropriate security parameters to ensure security against adversaries with certain concrete capabilities.

Bellare and Rogaway ([BR96]) argue that, in order to be able to deduce concrete security recommendations, it is important to be precise in the reduction from a forger to the algorithm that solves the hard problem. For example, if one knows that factoring integers of length l is no more than 100 times harder than breaking a certain signature scheme with security parameter l, then one could pick l so that even 1% of the work required to factor integers of length l is considered infeasible.

A reduction in which the difficulty of forging and the difficulty of solving the underlying hard problem are close is called *tight*; otherwise, it is called *loose*. (Naturally, "close," "tight" and "loose" are imprecise terms and make more sense when used in the comparative.) A scheme whose exact security is tightly related to the difficulty of factoring is also proposed in [BR96].

[*]MIT LCS, 545 Technology Square, Cambridge MA 02139 USA, `reyzin@mit.edu`, `http://theory.lcs.mit.edu/~reyzin`. This material is based upon work supported in part under a National Science Foundation Graduate Fellowship. A preliminary extended abstract of this work appears in [MR99].

1.2 Loose Security of Fiat-Shamir-like Signature Schemes

A fruitful method for constructing signature schemes was introduced by Fiat and Shamir ([FS86]). Although claimed for a specific ID scheme, the method works with a general *commit-challenge-respond* ID scheme. The method consists of replacing the verifier's random challenge by a publicly known "random" function H computed on the prover's commitment and the message being signed. This removes interaction and adds the message into the picture, thus changing an ID scheme into a signature scheme.

Many of such signature schemes have been proven secure when the "random" function is modeled as a random oracle ([BR93] provide a formal treatment of this model). However, the reductions in these proofs are quite loose, thus necessitating larger key sizes. Unless a tighter reduction has been overlooked, the only way to improve the security of such signature schemes is to modify them to allow for tighter reductions.

1.3 Tightening the Security of Fiat-Shamir-like schemes

This paper's main contribution is a modification to the factoring-based Fiat-Shamir-like schemes that makes their security very tightly related to the problem of integer factorization. Our modification is quite general and can be applied, in particular, to the schemes from [FS86], [FFS88], [OO88], [MS88], [OS90], [Oka92], [Mic94], [Sho96] and [Sch96].

To exemplify our method and make the description concrete, we picked one of the simpler and more efficient schemes from the above list, the one of [Mic94]. We shall henceforth call it "MSA" (for "Micali's signature algorithm"). We first present an exact analysis of the loose security of MSA, then propose the modification (called the "swap method") and present an exact analysis of the tight security of the modified scheme (called "MSA-swap"). Both MSA and MSA-swap are quite practical, with the performance comparable to that of the schemes currently used in practice.

1.4 When to Use Tight Security?

As Bellare and Rogaway explain in [BR96] using the schemes Full-Domain-Hash-RSA and PSS as examples, tighter exact security results in smaller security parameters and, hence, higher efficiency. Because Full-Domain-Hash-RSA and PSS have about the same running time for a given security parameter, PSS, with its tighter security, is better that Full-Domain-Hash-RSA.

Since then, folklore has often regarded schemes with tight security as superior to those with loose security, misunderstanding the examples of [BR96]. Indeed, an obvious (but often forgotten) observation is that

a more efficient scheme with loose security may be better than a less efficient one with tight security,

because the efficiency of the loosely-secure scheme can compensate for the higher security parameter.

To demonstrate this with a quantitative example, in Section 5 we analyze MSA against MSA-swap and PRab (a variant of PSS, also from [BR96], based on squaring rather than RSA). We find that the loosely-secure but more efficient MSA sometimes delivers more security for less cost than tightly secure schemes MSA-swap and PRab.

To our knowledge, despite the fact that our analysis is quite straightforward, no one has carried it out before (the paper [BR96] compares Full-Domain-Hash RSA and PSS, but the analysis is simpler because the two schemes have the same efficiency). The fact that a Fiat-Shamir-like scheme may be preferable to Rabin-based PRab for *both* security and efficiency reasons has not been pointed out before, either.

Our technique can be used to choose from among several signature schemes for a particular application. In particular, it can help one decide whether to use a Fiat-Shamir-like scheme directly or to apply our modification.

2 Definitions

Notation. We denote by $A^?$ a (probabilistic) oracle-calling algorithm; A^O denotes the same algorithm making calls to the specific oracle O, where $O : \{0,1\}^* \to \{0,1\}$. Because oracles returning more than one bit per question, as well as multiple oracles, can all be easily simulated by a single oracle returning one bit per question, we slightly abuse this notation and speak of, for example, $A^{?,?}$ and $A^{G,H}$, for some $G : \{0,1\}^* \to \{0,1\}^k$ and $H : \{0,1\}^* \to \{0,1\}^l$. Additionally, in some cases we need to allow one probabilistic oracle algorithm $A^?$ access to another probabilistic oracle algorithm $B^?$ with the oracle O; we denote it by A^{B^O}. Note that because $B^?$ is probabilistic, it is not, strictly speaking, an oracle; rather, this notation implies that, for each query, B^O gives a single answer selected, with the appropriate probability, from the set of all possible answers. When convenient and clear, we omit the superscript when speaking about $A^?$ or A^O.

We denote by $x \leftarrow A$ the fact that the probabilistic algorithm A output x.

Signature Schemes. Our definition of a signature scheme follows the one found in [GMR88] and refined in [BR93] to allow for random oracles. As is common, we decouple the notions of a signature scheme and of its security.

Definition 1 *A signature scheme with an oracle is a triple of probabilistic oracle algorithms* $\Pi = (Gen^?, Sign^?, Ver^?)$. *Gen is a key generation algorithm that, given the security parameter k, outputs a key pair* (pk, sk).[1] *Commonly, pk is called a* public key *and sk a* secret key *(sk is sometimes also called a* private key*). Sign is a signing algorithm: given a message M and a secret key sk it returns a signature x. Ver is a verification algorithm: given a public key pk, a message M and a purported signature x, it outputs "accept" or "reject."*

A signature x on a message M is called valid *with respect to a public key pk and an oracle H if* $Ver^H(pk, M, x) =$ *"accept." The only required relationship between the three algorithms is that the algorithm Sign output valid signatures: that is, for any oracle H and message M, if* $(pk, sk) \leftarrow Gen^H(1^k)$ *and* $x \leftarrow Sign^H(sk, M)$, *then x is a valid signature on M with respect to pk and H.*

Signature schemes with more than one security parameter can be defined similarly.

Note that in the above definition, we do not specify what the oracle H is or what it means for a signature scheme to be secure. This is discussed below. Our definition of security is a modified version of that in [BR96], which is based on [BR93] and [GMR88]. This definition concerns itself with exact, rather than asymptotic, security.

Intuitively, we want to capture the following in our definition of security: there is no algorithm (called "forger") that, for a random oracle H, is able to produce new valid signatures with reasonable probability in reasonable time without knowing sk. Moreover, we should assume that an attacker can coerce the signer into signing some number of messages of the attacker's choice—to carry out the so-called "adaptive chosen-message attack" [GMR88]. We model this by giving the forger oracle access to the oracle H and to the algorithm $Sign^H(sk, \cdot)$.

Definition 2 *A* forger $F^{?,?}$ *is a probabilistic two-oracle algorithm that is given a security parameter k and a public key pk as input. The first oracle of F is called a* hashing oracle *and the second oracle is called a* signature oracle. *Let H be a hashing oracle, and let* $(pk, sk) = Gen^H(1^k)$ *for some k. We say that the forger* succeeds *if* $(M, x) \leftarrow F^{H, Sign^H(sk,\cdot)}(1^k, pk)$ *and x is a valid signature on M with respect to pk and H, and F did not query its signature oracle on M.*

We say that a forger $(t, q_{sig}, q_{hash}, \varepsilon, \delta)$*-breaks the signature scheme if, for a security parameter k, the following holds:*

[1]Usually, we are interested in the running time of *Gen* as a function of k rather than $\log k$. Therefore, technically, we need to think of *Gen* as being given k in unary notation. This is denoted by 1^k.

- *its running time (plus the size of its description) does not exceed $t(k)$*
- *the number of its queries to the signature oracle does not exceed $q_{sig}(k)$*
- *the number of its queries to the hashing oracle does not exceed $q_{hash}(k)$*
- *with probability at least $\delta(k)$, $Gen^H(1^k)$ generates such a key (pk, sk) that the probability of the forger's success on input $(1^k, pk)$ is at least $\varepsilon(k)$ (here, the probability of the forger's success is taken over a random choice of the oracle H, the random tape of the forger, the random tape of the signer to whom the forger addresses the chosen-message queries, but not the choice of pk)*

Finally, we say that a signature scheme is $(t, q_{sig}, q_{hash}, \varepsilon, \delta)$-secure if no forger $(t, q_{sig}, q_{hash}, \varepsilon, \delta)$-breaks it.

(As an aside for the reader familiar with the definition of [BR96], we point out that if a scheme is $(t, q_{sig}, q_{hash}, \varepsilon\delta)$-secure in the sense of the [BR96], then it is $(t, q_{sig}, q_{hash}, \varepsilon, \delta)$-secure in the sense of the above definition. We simply separate the component of the probability that is due to the selection of the public key. This separation allows us to correct some minor errors in the security analysis present in the prior literature. See the proof of Theorem 2 for details.)

Measuring Signature Scheme Security. Now that we have defined what it means for a signature scheme to be secure, how do we actually prove anything about security? To do so, we relate the security of a signature scheme to the difficulty of some problem; in our case, the difficulty of factoring.

Definition 3 *Let $Gen(1^l)$ be an algorithm generating l-bit products of two primes. We say that an algorithm A (t, ε, δ)-factors integers generated by Gen if, for a given parameter l,*

- *A's running time (plus the size of its description) does not exceed $t(l)$*
- *with probability at least $\delta(l)$, $Gen(1^l)$ generates such an integer n that A has at least $\varepsilon(l)$ probability (taken over only the random choices of the algorithm, not the choice of n) of producing the correct factors of n on input n*

We say that factoring integers generated by Gen is (t, ε, δ)-secure if no such A exists.

Given this definition of the difficulty of a problem, we can then explain the security of a signature scheme Π in the following terms, as suggested by [BR96]: if some problem is $(t', \varepsilon', \delta')$-secure, then Π scheme is $(t, q_{sig}, q_{hash}, \varepsilon, \delta)$-secure. If t is not much smaller than t' and ε and δ are not much larger than ε' and δ', even for a reasonably large q_{sig} and q_{hash}, then the reduction proving the security is called *tight*.

3 Micali's Signature Algorithm (MSA)

3.1 Signature and Verification Algorithms

We describe the following ID and signature scheme from [Mic94], with similarities to the Ong-Schnorr ([OS90]) and the Guillou-Quisquater ([GQ88]) schemes.

Number Theory. Let k and l be two security parameters. Let $p_1 \equiv 3 \pmod 8$ and $p_2 \equiv 7 \pmod 8$ be two primes of approximately equal size and $n = p_1 p_2$ be an l-bit integer (such n is called a *Williams* integer [Wil80]). To simplify further computations, we will assume not only that $n > 2^{l-1}$, but also that $|Z_n^*| = n - p_1 - p_2 + 1 > 2^{l-1}$, and that $p_1 + p_2 - 1 < 2^{l/2+1}$. Let Q denote the set of non-zero quadratic residues modulo n. Note that $|Q| > 2^{l-3}$. Note also that for $x \in Q$, exactly one of its four square roots is also in Q (this follows from the fact that -1 is a non-square modulo p_1 and p_2 and the Chinese remainder theorem). Thus, squaring is a permutation over Q. From now on, when we speak of "the square root of x," we mean the single square root in Q; by $x^{2^{-k}}$ we will denote the single $y \in Q$ such that $x = y^{2^k}$. Also note that 2 is a non-square modulo p_1 and a square modulo p_2 (because $\left(\frac{2}{p}\right) = (-1)^{(p^2-1)/8}$), so $1 \in Q$ and $-1, 2, -2 \notin Q$. In general, for any $x \in Z_n^*$, exactly one of $x, -x, 2x, -2x$ is in Q.

Following [GMR88], define $F_0(x) = x^2 \bmod n$, $F_1(x) = 4x^2 \bmod n$, and, for an m-bit binary string $\sigma = b_1 \dots b_m$, define $F_\sigma : Q \to Q$ as $F_\sigma(x) = F_{b_m}(\dots(F_{b_2}(F_{b_1}(x)))\dots) = x^{2^m} 4^\sigma \bmod n$ (note that 4^σ is a slight abuse of notation, because σ is a binary string, rather than an integer; what is really meant here is 4 raised to the power of the integer represented in binary by σ). Because squaring is a permutation over Q and $4 \in Q$, F_σ is a permutation over Q.

Note that $F_\sigma(x)$ can be efficiently computed by anybody who knows n. Also, if one knows p_1 and p_2, one can efficiently compute $x = F_\sigma^{-1}(y)$ (as shown by Goldreich in [Gol86]) by computing $s = 1/4^{2^{-|\sigma|}} \bmod n$ and then letting $x = y^{2^{-|\sigma|}} s^\sigma \bmod n$ (these calculations can be done modulo p_1 and p_2 separately, and the results combined using the Chinese remainder theorem). However, if one does not know p_1 and p_2, then F_σ^{-1} is hard to compute, as shown in the Lemma below.

Lemma 1 *If one can compute, for a given $y \in Q$ and two different strings σ and τ of equal length, $x_1 = F_\sigma^{-1}(y)$ and $x_2 = F_\tau^{-1}(y)$, then one can factor n.*

Proof The proof is by induction on the length of the strings σ and τ.

If $|\sigma| = |\tau| = 1$, then assume, without loss of generality, that $\sigma = 0$ and $\tau = 1$. Then $F_0(x_1) \equiv F_1(x_2) \equiv y \bmod n$, i.e., $x_1^2 \equiv 4x_2^2 \bmod n$, i.e., $n|(x_1 - 2x_2)(x_1 + 2x_2)$. Note that $x_1, x_2 \in Q$ and $\pm 2 \notin Q$, so $\pm 2x_2 \notin Q$, so $x_1 \not\equiv \pm 2x_2 \pmod{n}$, so n does not divide either $x_1 - 2x_2$ or $x_1 + 2x_2$. Thus, by computing the gcd of $x_1 + 2x_2$ and n, we can get either p_1 or p_2.

For the inductive case, let σ and τ be two strings of length $m+1$. Let σ' and τ' be their m-bit prefixes, respectively. If $F_{\sigma'}(x_1) \equiv F_{\tau'}(x_2) \pmod{n}$, we are done by the inductive hypothesis. Otherwise, the last bit of σ must be different from the last bit of τ, so, without loss of generality, assume the last bit of σ is 0 and the last bit of τ is 1. Then $F_0(F_{\sigma'}(x_1)) \equiv F_1(F_{\tau'}(x_2)) \pmod{n}$, and the same proof as for the base case works here. ■

THE ID SCHEME. The above lemma naturally suggests the following ID scheme. A user has n as the public key and p_1, p_2 as the secret key. To prove his identity (i.e., that he knows p_1 and p_2) to a verifier, he commits to random $X \in Q$ and sends it to the verifier. The verifier produces a random k-bit challenge σ and sends it to the prover (the user). The prover responds with $z = F_{0\sigma}^{-1}(X)$ (note that σ here is prefixed with a single 0 bit, whose use will be explained shortly). The verifier checks that $X = F_{0\sigma}(z) = F_\sigma(z^2)$ and that $z \not\equiv 0 \pmod{n}$. Informally, the security of this protocol is based on the fact that if the prover is able to respond to two different challenges σ and τ, then, by Lemma 1, he knows p_1 and p_2. The 0 bit in front of σ is to save the verifier from having to check that the prover's response is in Q (which is a hard problem in itself)—instead, she just squares the prover's response and thus puts it in Q.

We will say no more about the security of this ID scheme because we are not concerned with it in this paper. We will, however, point out an efficiency improvement for the prover. First, as part of key generation, the prover computes, using the Chinese remainder theorem, $s = 1/4^{2^{-k-1}} \bmod n$. Then, when committing to a random $X \in Q$, the prover randomly selects an $x \in Z_n^*$ and sets $X = x^{2^{k+1}} \bmod n$ (note that X gets selected with uniform distribution as long as x is so selected). Now, to respond to a challenge σ, the prover simply computes $z = xs^\sigma \bmod n$.

MSA. The standard way to change the above ID scheme into a signature scheme is to replace the verifier with a random function $H : \{0,1\}^* \to \{0,1\}^k$. The exact steps of the algorithms *Gen*, *Sign* and *Ver* follow.

Key Generation

1. Generate two random primes $p_1 \equiv 3 \pmod 8$ and $p_2 \equiv 7 \pmod 8$ and $n = p_1 p_2$ so that $n < 2^l$, $n - p_1 - p_2 + 1 > 2^l + 1$ and $p_1 + p_2 - 1 < 2^{l/2+1}$
2. Generate coefficient $c = {p_2}^{-1} \bmod p_1$ for use in the Chinese remainder theorem
3. Compute $u_i = \left(\frac{p_i+1}{4}\right)^{k+1} \bmod \frac{p_i-1}{2}$ for $i = 1, 2$ (note that u_i is such that raising a square to the power u_i modulo p_i will compute its 2^{k+1} root)
4. Compute $s_i = \left(\frac{p_i+1}{4}\right)^{u_i} \bmod p_i$ for $i = 1, 2$ (this computes $1/4^{2^{-(k+1)}} \bmod p_i$)
5. Compute $v = (s_1 - s_2)c \bmod p_1$ and $s = s_2 + vp_2$ to get $s = 1/4^{2^{-(k+1)}} \bmod n$
6. Output n as the public key and (n, s) as the secret key (to make signing more efficient, p_1, p_2, s_1, s_2 and c should be added to the secret key, in which case s is not needed and the previous step can be omitted)

Signing

1. Generate X by picking a random $x \in Z_n^*$ and computing $X = x^{2^{k+1}} \bmod n$ (note that this step can be done off-line, before the message is known)
2. Compute $\sigma = H(X, M)$, and $z = F_{0\sigma}^{-1}(X)$ via $t = s^\sigma \bmod n$ (this can be done via $t_i = s_i^\sigma \bmod p_i$ for $i = 1, 2$, $v = (t_1 - t_2)c \bmod p_1$, $t = t_2 + vp_2$) and $z = xt \bmod n$
3. Output (z, σ)

Verifying

1. Verify that $z \not\equiv 0 \pmod n$ and compute $X = F_\sigma(z^2)$ via $t_1 = z^{2^{k+1}} \bmod n$, $t_2 = 2^\sigma \bmod n$, $X = t_1 t_2 \bmod n$
2. Verify if $\sigma = H(X, M)$

3.2 Security of MSA

We state the following two theorems that give two different views of the exact security of MSA and demonstrate the tradeoff between running time and success probability. Their proofs use known methods (see Pointcheval and Stern [PS96] and Ohta and Okamoto [OO98]). Our probability analysis is new, however, and results in slightly tighter reductions.

Theorem 1 *If there exists a forger that $(t, q_{sig}, q_{hash}, \varepsilon, \delta)$-breaks MSA with security parameters l and k, then there exists an algorithm that $(t', \varepsilon', \delta)$-factors integers generated by Gen for*

$$\begin{aligned} t' &= 2t + 2(q_{sig}+1)T_1 + T_2 \\ \varepsilon' &= \frac{(\varepsilon-\gamma)^2}{q_{hash}+1} - 2^{-k}(\varepsilon - \gamma), \end{aligned}$$

where T_1 is the time required to perform an MSA signature verification, T_2 is the time required to factor n given the conditions of Lemma 1 (essentially, a gcd computation) and $\gamma = 2^{-l+3} q_{sig}(q_{hash}+1)$ (note that γ is close to 0 for a large enough l).

Proof

Main Idea. Let F be a forger that $(t, q_{sig}, q_{hash}, \varepsilon, \delta)$-breaks MSA. We will construct a factoring algorithm A that uses F to produce $y, z \in Z_n^*$ and $\sigma \neq \tau \in \{0,1\}^k$ such that $F_\sigma(z^2) = F_\tau(y^2)$. This will allow A to factor n by using the method given in the proof of Lemma 1.

The main idea of this proof is given by the "forking lemma" of [PS96]. It is to allow F to run once to produce one forgery—a signature (z, σ) on a message M such that $\sigma = H(X, M)$ where $X = F_\sigma(z^2)$. Note that F had to ask a hashing-oracle query on (X, M)—otherwise its probability of success is at most 2^{-k}. Then, run F the second time, giving the same answers to all the oracle queries before the query (X, M). For (X, M) give a new answer τ. Then, if F again forges a signature (y, τ) using X and M, we will have achieved our goal.

Assuming n is such that F has probability at least ε of success, the probability that A will factor n using this approach is roughly ε^2/q_{hash}, because F needs to succeed twice and we have no guarantee that F will choose to use (X, M) for its second forgery and not any of its other q_{hash} oracle queries.

Details. We will now provide the details of the construction. We first have to show precisely how A interacts with F. We will assume that F performs the necessary bookkeeping and does not ask the same hash query twice.[2] Note that F may ask the same signature query twice, because the answers will most likely be different. We will also modify F so that it will not output a signature (z, σ) on a message M unless it knows that the signature is correct; in particular, that means that F has to first ask, in a hash query, for the value of $H(F_\sigma(z^2), M)$. This may increase the number of hash queries by one and the running time of F by the cost of one signature verification.

A first comes up with a random tape for F, remembers it, and runs F on that tape. A maintains two tables: a signature query table and a hash query table.

In order to answer a signature query on a message M'_j, A comes up with a random $z_j \in Q$ and $\sigma'_j \in \{0,1\}^k$, computes $X'_j = F_{\sigma'_j}(z_j^2)$, and checks its signature query table to see if a signature query on M'_j has already been asked and X'_j used in answering it. If so, A changes z_j and σ'_j to the z and σ that were used in answering that query. Then A adds the entry $(j, z_j, \sigma'_j, X'_j, M'_j)$ to its signature query table and outputs (z_j, σ'_j).

In order to answer the i-th hashing query (X_i, M_i), A first checks its signature query table to see if there is an entry $(j, z_j, \sigma'_j, X'_j, M'_j)$ such that $(X'_j, M'_j) = (X_i, M_i)$. If so, it just outputs σ'_j. Otherwise, it picks a random $\sigma_i \in \{0,1\}^k$, records in its hash query table the quadruple (i, X_i, M_i, σ_i) and outputs σ_i.

Assume now that F outputs a signature (z, σ) on a message M. Let $X = F_\sigma(z^2)$. Because we modified F to first ask a hash query on (X, M), we have that, for some i, $(X, M, \sigma) = (X_i, M_i, \sigma_i)$ in the hash query table (it can't come from the signature query table, because F is not allowed to forge a signature on a message for which it asked a signature query). A finds such an i in its table and remembers it.

A now runs F on the same random tape as the first time, giving the exact same answers to all F's queries before the i-th hash query (it can do so because it has all the answers recorded in the tables). Note that this means that F will be asking the same i-th hash query (X_i, M_i) as the first time. As soon as F asks the i-th hash query, however, A stops giving the answers from the tables and comes up with new answers at random, in the same manner as the first time. Let τ_j be the answer given to the j-th hash query for $j \geq i$.

Assume now that F again outputs a signature (y, τ) on message M', and let $X' = F_\tau(y^2)$. We know that F had to ask a hash query on (M', X'). If it was the i-th hash query, then $X' = X = X_i$, so $F_\sigma(z^2) = F_\tau(y^2)$. Thus, as long as $\sigma \neq \tau$, A can factor n by essentially performing a gcd computation (described in the proof Lemma 1).

Probability Analysis. First, we need the following lemma.

Lemma 2 *Let $a_1, a_2, \ldots, a_\lambda$ be real numbers. Let $A = \sum_{\mu=1}^{\lambda} a_\mu$. Let $S = \sum_{\mu=1}^{\lambda} {a_\mu}^2$. Then $S \geq \frac{A^2}{\lambda}$.*

Proof Let $a = A/\lambda$ and $b_\mu = a - a_\mu$. Note that $\sum_{\mu=1}^{\lambda} b_\mu = \lambda a - \sum_{\mu=1}^{\lambda} a_\mu = A - A = 0$. Then

$$\sum_{\mu=1}^{\lambda} {a_\mu}^2 = \sum_{\mu=1}^{\lambda} (a - b_\mu)^2 = \lambda a^2 - 2a \sum_{\mu=1}^{\lambda} b_\mu + \sum_{\mu=1}^{\lambda} b_\mu^2 \geq \lambda a^2 = \frac{A^2}{\lambda}.$$

[2]This may slightly increase the running time of F, but we will ignore costs of simple table look-up for the purposes of this analysis.

■

For this analysis, we assume that n is such that F has probability at least ε of success. By assumption, such n is generated with probability at least δ.

Consider a single run of F, with queries answered by A. Because F is interacting with A rather than with the true oracles that F expects, we need to compare the distribution of A's answers to the distribution of answers given by the true oracles. The distributions differ only as follows: if, for some signature query j, the hash value of (X'_j, M'_j) has already been defined through a previous answer to a hash query, then A will fail, whereas the true oracles would not. The probability ε of F's success is thus reduced by the probability that A will fail, which we shall now bound. Note that z_j is picked by A at random from Q, so $X'_j = F_{\sigma'_j}(z_j^2)$ is a random element of Q. There are q_{sig} signature queries and $q_{hash}+1$ hash queries. Thus, the probability that there exist such signature query and hash query is at most $q_{sig}(q_{hash}+1)/|Q| \le q_{sig}(q_{hash}+1)2^{-l+3} = \gamma$. Thus, the probability of F's success when interacting with A rather than the true oracles is at least $\varepsilon - \gamma$.

We will now calculate the probability of the event that F outputs a valid forgery based on the same hash query both times and that the hash query was answered differently the second time. Let p_j be the probability that, in one run, F produces a valid forgery based on hash query number j. Clearly,

$$\varepsilon - \gamma \le \sum_{j=1}^{q_{hash}+1} p_j.$$

Let $p_{j,s}$ (for a sufficiently long binary string s of length m) be the probability that, in one run, F produces a valid forgery based on hash query number j given that the string s was used to determine the random tape of F and the responses to all the queries of F until (and not including) the j-th hash query. We have that

$$2^m p_j = \sum_{s\in\{0,1\}^m} p_{j,s}.$$

Given such a fixed string s, the probability that F produces a valid forgery based on the hash query number j in both runs is $p_{j,s}^2$ (because the first forgery is now independent of the second forgery). The additional requirement that the answer to the hash query in the second run be different reduces this probability to $p_{j,s}(p_{j,s} - 2^{-k})$. Thus, the probability q_j that F produces a valid forgery based on the hash query number j in both runs and that the answer to the hash query is different in the second run is

$$q_j = \sum_{s\in\{0,1\}^m} 2^{-m} p_{j,s}(p_{j,s} - 2^{-k}) = 2^{-m}\left(\sum_{s\in\{0,1\}^m} p_{j,s}^2 - 2^{-k}\sum_{s\in\{0,1\}^m} p_{j,s}\right) \ge \frac{2^{-m}\left(p_j 2^m\right)^2}{2^m} - 2^{-k}p_j = p_j^2 - 2^{-k}p_j$$

(by Lemma 2).

The probability that F outputs a valid forgery based on the same hash query both times and that the hash query was answered differently in the second run is now

$$\sum_{j=1}^{q_{hash}+1} q_j \ge \sum_{j=1}^{q_{hash}+1} p_j^2 - \sum_{j=1}^{q_{hash}+1} 2^{-k}p_j \ge \frac{(\varepsilon-\gamma)^2}{q_{hash}+1} - 2^{-k}(\varepsilon - \gamma)$$

(by Lemma 2). ■

Theorem 2 *If there exists a forger that $(t, q_{sig}, q_{hash}, \varepsilon, \delta)$-breaks MSA with security parameters l and k, such that $\varepsilon > 2^{-k+1}(q_{hash}+1) + \gamma$, then there exists an algorithm that $(t', \varepsilon', \delta)$-factors integers generated by Gen for*

$$\begin{aligned} t' &= \frac{(2q_{hash}+3)(t + q_{sig}T_1)}{\varepsilon - \gamma - 2^{-k+1}(q_{hash}+1)} + T_2 \\ \varepsilon' &= \frac{1}{2}\left(1 - \frac{1}{e}\right)^2 > 0.199, \end{aligned}$$

where T_1, T_2 and γ are as in Theorem 1.

Proof

Main Idea. The idea is to iterate the A from Theorem 1 sufficiently many times to get a constant probability of success. More specifically, run F about $1/\varepsilon$ times the first time, to achieve a constant probability of a successful forgery, and about $2q_{hash}/\varepsilon$ times the second time, to achieve a constant probability of a successful forgery that uses the pair (X, M).

Details. A here works very similar to the A of Theorem 1. We will make exact same assumptions on F and modifications to it. For the first run, A simply repeats the procedure if F does not succeed, up to $(\varepsilon - \gamma)^{-1}$ times. Every repetition starts entirely anew—with a new random tape for F and new random answers from A (note, though, that the public key that is input to F is always the same).

Assuming F succeeds, A moves on to the second run. The second run is repeated until F succeeds in producing a forgery on the i-th query, up to

$$\frac{2q_{hash}+2}{\varepsilon-\gamma-2^{-k}(2q_{hash}+2)}$$

times. Every repetition starts with the same random tape for F and the same answers for queries up to the i-th query; the answers to the i-th query and beyond are generated anew at random.

Probability Analysis. A similar analysis to what appears below has been carried out in [OO98] and [PS00]. We wish to point out one crucial difference, however: both [OO98] and [PS00] assume, in their probability analyses, that the $1/\varepsilon$ first runs of F are entirely independent. This is incorrect, because the public key input to F is always the same. Indeed, if F is such that it never succeeds on a certain public key, then running it repeatedly will not increase the success probability. Therefore, it is necessary to separate the component of the probability that is due to the selection of the public key, as we do in our definition of security.

We need the following lemma for this analysis. It is a restatement in probabilistic terms of the "heavy row lemma," found, in various forms, in [OO98], [PS96] and [FFS88], among others.

Lemma 3 *Let E be an event with probability α. Let $E_1, \ldots, E_\lambda$ be disjoint events such that $\sum_{\mu=1}^{\lambda} \Pr[E_\mu] = 1$. Let ξ, $1 \le \xi \le \lambda$ be a random variable with the following distribution:* $\Pr[\xi = \mu] = \Pr[E_\mu|E]$. *Then*

$$\Pr_{\xi}\left[\Pr\left[E|E_\xi\right] > \frac{\alpha}{2}\right] \ge \frac{1}{2}.$$

Informally, this lemma can be stated as follows. Think of E as the event of an experiment being successful, and each E_μ as the event of the experiment having certain preconditions. The probability of the experiment's success (regardless of preconditions) is α. The lemma says that if the experiment actually succeeded, then it is quite likely (probability at least 1/2) its preconditions were such that, if repeated with the same preconditions, its probability of success is at least $\alpha/2$. That is, if the experiment succeeded, probably its preconditions were not too bad.

Proof In two sentences, the proof is simply that the set of preconditions that contribute only $\frac{\alpha}{2}$ to the success of E cannot together contribute more than a half of all the successes of E. Thus, the other half of successes has to come from preconditions which contribute more than $\frac{\alpha}{2}$ to the success of E.

More formally, suppose $\Pr_\xi\left[\Pr\left[E|E_\xi\right] > \frac{\alpha}{2}\right] < \frac{1}{2}$. Then $\Pr_\xi\left[\Pr\left[E|E_\xi\right] \le \frac{\alpha}{2}\right] > \frac{1}{2}$. Let $S = \{\mu | Pr[E|E_\mu] \le \frac{\alpha}{2}\}$, and let $E_S = \bigvee_{\mu \in S} E_\mu$. Then $\Pr_\xi[\xi \in S] > \frac{1}{2}$. But also $\Pr_\xi[\xi \in S] = \Pr[E_S|E] = \frac{\Pr[E_S \wedge E]}{\Pr[E]} = \frac{\sum_{\mu\in S}\Pr[E\wedge E_\mu]}{\alpha} = \frac{\sum_{\mu\in S}\Pr[E|E_\mu]\Pr[E_\mu]}{\alpha} < \frac{\sum_{\mu\in S}(\alpha/2)\Pr[E_\mu]}{\alpha} \le \frac{1}{2}$, which is a contradiction. ■

For this analysis, we assume that n is such that F has probability at least ε of success. By assumption, such n is generated with probability at least δ. (This assumption on n is necessary, because multiple runs of F are not entirely independent—they have the same n as input to F. For example, if n is such that F never succeeds for it, then no amount of repetition will increase the probability of success. This is why our definition of security separates the component of the probability that is due to the selection of the public key.)

It will be easier to carry out the analysis if we first replace A with A' that operates similarly to A, except for the following. In the first part of the algorithm, it picks a query number i ($1 \le i \le q_{hash}+1$) at random and a string s at random for the random tape of F and answers to all its queries, up to the i-th hash query. Starting from the i-th hash query, it answers queries at random (rather than from the string s). A' considers a run successful only if the forgery is based on the i-th hash query.

Note that interacting with A (or A') instead of interacting with the true oracles has the same impact on F's probability of success as in the proof of Theorem 1: it reduces it by at most γ. The probability that F succeeds in producing a forgery in a single run based on the i-th hash query is at least $\alpha = (\varepsilon-\gamma)/(q_{hash}+1)$. If we now repeat the first run α^{-1} times (as long as $\alpha > 0$), the probability of a successful forgery is at least

$$1-(1-\alpha)^{\alpha^{-1}} \ge 1-1/e$$

(where e is the base of the natural logarithm).

Assume that F did produce a successful forgery, based on hash query number i and string s. Let $p_{i,s}$ be the probability that, if i and s are picked as above, F produces a successful forgery based on hash query number i and string s. Note that $p_{i,s}$ is exactly the probability of a successful forgery on a second run. Also note that, by Lemma 3, $\Pr[p_{i,s} \ge \alpha/2] \ge 1/2$.

Assume now that indeed $p_{i,s} \ge \alpha/2$. Subtract from it the probability that σ and τ are the same (in that case, just like in the proof of Theorem 1, A does not have enough information to factor). If we now repeat the second run of F $\left(\alpha/2-2^{-k}\right)^{-1}$ times (as long as $\alpha/2 > 2^{-k}$), the probability that A' will be able to factor n is at least

$$1-\left(1-\left(\alpha/2-2^{-k}\right)\right)^{\left(\alpha/2-2^{-k}\right)^{-1}} \ge 1-1/e.$$

Multiplying together the probability of success in the first run, the probability that $p_{i,s} \ge \alpha/2$ and the probability of success in the second run, we get that the probability of success of A' is at least

$$\frac{(1-1/e)^2}{2} \ge 0.199.$$

Now replace A' with A to obtain a slight decrease in running time (by a factor of about 1.5) without decreasing the probability of success. The difference is that, in the first part of the algorithm, A does not try to guess the number i of the query on which F will succeed, but rather accepts F's success on any query. This increases the success probability of a single run by a factor of $(q_{hash}+1)$ and thus decreases the running time of the first part by the same factor. We only need to show that this does not affect the success probability of the second part of the algorithm.

The only difference for the second part of the algorithm is in how the query number i and the string s are determined. We will show that the distribution of the pairs (i, s) is the same for A as for A'. Intuitively, this is so because nothing changes in the distribution of (i, s) simply because A' throws away, at random, successful forgeries by F when they don't agree with its choice of i. Indeed, in the case of A', for a given pair (i, s) the probability that it will be used in the second part is

$$\begin{aligned} \Pr[i \text{ and } s \text{ are picked by } A' \mid F \text{ forges on the query that is picked by } A'] &= \\ \frac{\Pr[i \text{ and } s \text{ are picked and } F \text{ forges on query } i]}{\Pr[F \text{ forges on the query that is picked by } A']} &= \end{aligned}$$

$$\frac{\Pr[F \text{ forges on query } i \text{ and } s \text{ is picked}]}{(q_{hash}+1)\Pr[F \text{ forges on the query that is picked by } A']} =$$
$$\frac{\Pr[F \text{ forges on query } i \text{ and } s \text{ is picked by } A \text{ for random tape and answers up to query } i]}{\Pr[F \text{ forges}]} =$$
$$\Pr[i, s \text{ come out of the first run of } A \mid F \text{ forges}],$$

which is exactly the probability that (i, s) is picked in the case of A.

Running Time Analysis. The running time of the first part is $(\varepsilon - \gamma)^{-1}(t + q_{sig}T_1)$. The running time of the second part is $\left(\alpha/2 - 2^{-1}\right)^{-1}(t + q_{sig}T_1)$. Adding in the time for the gcd computation, we get that the total running time is at most

$$\left(\frac{1}{\varepsilon-\gamma} + \frac{1}{(\varepsilon-\gamma)/2(q_{hash}+2) - 2^{-k}}\right)(t + q_{sig}T_1) + T_2 \geq$$
$$\left(\frac{1}{\varepsilon-\gamma-2^{-k+1}(q_{hash}+1)} + \frac{2q_{hash}+2}{\varepsilon-\gamma-2^{-k+1}(q_{hash}+1)}\right)(t + q_{sig}T_1) + T_2 =$$
$$\frac{(2q_{hash}+3)(t+q_{sig}T_1)}{\varepsilon-\gamma-2^{-k+1}(q_{hash}+1)} + T_2.$$

■

The following two statements follow directly from the theorems just proved once we fix the parameters to be high enough to avoid dealing with small terms.

Corollary 1 *If factoring l-bit integers generated by Gen is $(t', \varepsilon', \delta)$-secure, then MSA is $(t, q_{sig}, q_{hash}, \varepsilon, \delta)$-secure for*

$$\begin{aligned} t &= t'/2 - (q_{sig}+1)T_1 - T_2/2 \\ \varepsilon &= \sqrt{2\varepsilon'(q_{hash}+1)} + 2^{-k}(q_{hash}+1) + 2^{-l+3}q_{sig}(q_{hash}+1) \end{aligned}$$

Proof Note that the value for t follows directly by solving for t the equation for t' in the statement of Theorem 1. The value for ε is computed as follows: solve for $\varepsilon - \gamma$ the quadratic equation[3] that expresses ε' in terms of $\varepsilon - \gamma$ to get

$$\begin{aligned} \varepsilon - \gamma &= \left(2^{-k}(q_{hash}+1) + \sqrt{2^{-2k}(q_{hash}+1)^2 + 4\varepsilon'(q_{hash}+1)}\right)/2 \\ &\leq \left(2^{-k}(q_{hash}+1) + \sqrt{2^{-2k}(q_{hash}+1)^2} + \sqrt{4\varepsilon'(q_{hash}+1)}\right)/2 \\ &= 2^{-k}(q_{hash}+1) + \sqrt{\varepsilon'(q_{hash}+1)}. \end{aligned}$$

(Note the importance of careful analysis, which is due to [BM99]: a less careful, but seemingly harmless, approximation to the solution of the quadratic equation would result in $2^{-k}(q_{hash}+1)$ being replaced with $2^{-k/2}\sqrt{q_{hash}+1}$, thus requiring an increase in the security parameter k.) ■

Corollary 2 *If factoring l-bit integers generated by Gen is $(t', 0.199, \delta)$-secure, then MSA is $(t, q_{sig}, q_{hash}, \varepsilon, \delta)$-secure for*

$$t = \frac{(t'-T_2)\varepsilon}{4q_{hash}+6} - q_{sig}T_1$$

[3]Note the importance of careful analysis: a less careful, but seemingly harmless, approximation to the solution of the quadratic equation would result in $2^{-k}(q_{hash}+1)$ being replaced with $2^{-k/2}\sqrt{q_{hash}+1}$, thus requiring an increase in the security parameter k. This observation is made in [BM99].

as long as

$$\varepsilon \;\geq\; 2\left(2^{-k+1}(q_{hash}+1) + 2^{-l+3}q_{sig}(q_{hash}+1)\right).$$

Proof Similarly to Corollary 1, this corollary is simply the contrapositive of Theorem 2 with some low-order terms removed.

Solving for t the equation for t' of Theorem 2, we obtain

$$t = \frac{(t' - T_2)(\varepsilon - \gamma - 2^{-k+1}(q_{hash}+1))}{2q_{hash}+3} - q_{sig}T_1.$$

The condition on ε ensures that $\varepsilon - \gamma - 2^{-k+1}(q_{hash}+1) \geq \varepsilon/2$. Observe that we are allowed to decrease t, as this will only weaken the result, so we are allowed to replace $\varepsilon - \gamma - 2^{-k+1}(q_{hash}+1)$ by $\varepsilon/2$.

Note also that Theorem 2 requires that $\varepsilon \geq 2^{-k+1}(q_{hash}+1) + \gamma$, which is also assured here by the conditions on ε. ■

4 The Swap Method

4.1 Motivation

As exemplified by the proof of Theorems 1 and 2 above, all known results for the security of Fiat-Shamir-like signature schemes involve losing a factor of q_{hash} (in either time or success probability) in the reduction from a forger to an algorithm that breaks the underlying hard problem (see, for example, [FS86], [Sch96], [PS96], [Sho96], [OO98], [PS00]). While no proof exists that the loss of this factor is necessary, the problem seems inherent in the way signature schemes are constructed from ID schemes, as explained below.

The security of an ID scheme usually relies on the fact that a prover would be unable to answer two different challenges for the same commitment without knowing the private key. Therefore, in the proof of security of the corresponding signature scheme, we need to use the forger to get two signatures on the same commitment, as we did in the proofs of Theorems 1 and 2. The forger, however, has any of its q_{hash} queries to pick for the commitment for the second signature—hence, our loss of the factor of q_{hash}. We want to point out that q_{hash} is a significant factor, and its loss definitely makes a reduction quite loose. This is because a reasonable bound on the number of possible hash queries of committed adversaries is about $q_{hash} = 2^{80}$ (see Section 4.4).

We therefore devise a new method of constructing signature schemes from ID schemes so that any one signature from the forger is enough to break the underlying hard problem.

4.2 Method

Recall that in Fiat-Shamir-like signature schemes, the signer comes up with the commitment and then uses H applied to the commitment and the message to produce the challenge. We propose that instead the signer *first come up with the challenge and then use H applied to the challenge and the message to produce the commitment.* In a way, we swap the challenge and the commitment.

This method applies whenever the signer can compute the response given only the challenge and the commitment. It does not apply when information used during the generation of the commitment is necessary to compute the response. For example, it does not apply to discrete-logarithm-based ID schemes (such as the Schnorr scheme [Sch89]) in which the prover needs to know the discrete logarithm of the commitment in order to provide the response.

Additionally, in order to use this method, one needs to get around the problem that the commitment is selected from some structured set (such as Q in the case of MSA), while H returns a random binary string. This problem can usually be easily solved. The only case known to us when it seems to present a real obstacle is in the scheme of Ohta and Okamoto ([OO88]) in the case when an exponent L is used such that $\gcd(L, (p_1 - 1)(p_2 - 1)) > 2$.

The key-generation algorithm and the private key may need to be modified slightly in order to provide the signer with the additional information needed to compute the response from a random commitment, rather than from a commitment that it generated. The verification algorithm remains vastly unchanged.

In the next section, we exemplify our proposed method and explain why it results in a tighter security reduction.

4.3 MSA-swap

4.3.1 Description

The scheme depends on two security parameters: k and l. Let $H : \{0,1\}^* \to \{0,1\}^{l-1}$ be a random function.

Key Generation The key generation is the same as in MSA, except for one additional step (step 6) and extra information in the private key:

1. Generate two random primes $p_1 \equiv 3 \pmod 8$ and $p_2 \equiv 7 \pmod 8$ and $n = p_1 p_2$ so that $n < 2^l$, $n - p_1 - p_2 + 1 > 2^l + 1$ and $p_1 + p_2 - 1 < 2^{l/2+1}$
2. Generate coefficient $c = {p_2}^{-1} \bmod p_1$ for use in the Chinese remainder theorem
3. Compute $u_i = \left(\frac{p_i+1}{4}\right)^{k+1} \bmod \frac{p_i-1}{2}$ for $i = 1, 2$ (note that u_i is such that raising a square to the power u_i modulo p_i will compute its 2^{k+1} root)
4. Compute $s_i = \left(\frac{p_i+1}{4}\right)^{u_i} \bmod p_i$ for $i = 1, 2$ (this computes $1/4^{2^{-(k+1)}} \bmod p_i$)
5. Compute $v = (s_1 - s_2)c \bmod p_1$ and $s = s_2 + vp_2$ to get $s = 1/4^{2^{-(k+1)}} \bmod n$
6. If u_i is odd, make it even by setting $u_i = u_i + \frac{p_i-1}{2}$ for $i = 1, 2$ (note that now u_i is such that raising a square or its negative to the power u_i modulo p_i will compute its 2^{k+1} root)
7. Output n as the public key and $(n, s, u_1, u_2, p_1, p_2, c)$ as the secret key

Signing

1. Generate a random σ and compute $t = s^\sigma \bmod n$ (note that this step can be done off-line, before the message is known).
2. Compute $X = H(\sigma, M)$. If $X \notin Z_n^*$, then try another σ (the probability of that is at most $2^{-l/2+2}$). If the Jacobi symbol $\left(\frac{X}{n}\right) = -1$, set $X = 2X \bmod n$. Now either X or $n - X$ is in Q. Compute $z = F_{0\sigma}^{-1}(\pm X)$ via $x_i = X^{u_i} \bmod p_i$ for $i = 1, 2$, $v = (x_1 - x_2)c \bmod p_1$, $x = x_2 + vp_2$ and $z = xt \bmod n$.
3. Output (z, σ).

Verifying

1. Verify that $z \not\equiv 0 \pmod n$ and compute $X = F_\sigma(z^2)$ via $t_1 = z^{2^{k+1}} \bmod n$, $t_2 = 2^\sigma \bmod n$, $X = t_1 t_2 \bmod n$ (this step is the same as for MSA).
2. Let $X' = H(\sigma, M)$. If $X \equiv \pm X' \pmod n$ or $X \equiv \pm 2X' \pmod n$, accept the signature (this step differs slightly from MSA).

4.3.2 Security of MSA-swap

Theorem 3 *If there exists a forger that $(t, q_{sig}, q_{hash}, \varepsilon, \delta)$-breaks MSA-swap with security parameters l and k, then there exists an algorithm that $(t', \varepsilon', \delta)$-factors integers generated by Gen for*

$$\begin{aligned} t' &= t + 2(q_{sig} + q_{hash} + 1)T_1 + T_2 \\ \varepsilon' &= \varepsilon - \gamma, \end{aligned}$$

where T_1 is the time required to perform an MSA-swap signature verification, T_2 is the time required to factor n given the conditions of Lemma 1 (essentially, a gcd computation) and $\gamma = 2^{-k}q_{sig}(q_{hash}+1) + 2^{-l/2+2}(q_{hash} + q_{sig} + 1)$ (note that γ is close to 0 for large enough k and l).

Proof

Main Idea.

Familiarity with the proof of Theorem 1 will be helpful for understanding this proof.

Let F be a forger that $(t, q_{sig}, q_{hash}, \varepsilon, \delta)$-breaks MSA-swap. Similarly to the proof of Theorem 1, we will construct an algorithm that, after interacting with F, will produce $y, z \in Z_n^*$ and $\sigma \neq \tau \in \{0,1\}^k$ such that $F_\sigma(z^2) = F_\tau(y^2)$.

The main idea is to answer each hash query on (σ, M) with an X computed via $X = F_\tau(y^2)$ for a random $y \in Z_n^*$ and arbitrary τ that is different from σ. Then if F forges a signature (z, σ) on M, we will have $F_\sigma(z^2) = F_\tau(y^2)$ and will be able to factor n.

Details. Just like in the proof of Theorem 1, we make the assumption that F does not ask the same hash query twice and modify F to output only correct signatures—in particular, to ask a hash query on (σ, M) before outputting a signature (z, σ) on M.

A maintains two tables, for signature query answers and for hash query answers. To answer the j-th signature query on a message M'_j, A picks a random $\tau'_j \in \{0,1\}^k$ and checks its signature query table to see if a signature query on M'_j has already been asked and if τ'_j was used in answering it. If so, it returns the answer from the table. If not, it picks a random $y'_j \in Q$ and a random $\mu'_j \in \{1, -1, 2, -2\}$, computes $X'_j = F_{\tau'_j}(y'^2_j)/\mu'_j \bmod n$, and checks if $X'_j < 2^{l-1}$. If not, it restarts with new random y'_j and μ'_j. The expected number of trials is less than 2, because X'_j is a random element in Z_n^* (because $F_{\tau'_j}(y'^2_j)$ is a random element in Q and μ'_j is also random) and $|Z_n^*| < n < 2^l$. Once X'_j satisfies the desired condition, A outputs (y'_j, τ'_j) as the signature and records $(j, y'_j, \tau'_j, X'_j, M'_j)$ in its signature query table.

To answer the i-th hash query on (σ_i, M_i), A first checks if an entry $(j, y'_j, \tau'_j, X'_j, M'_j)$ such that $\sigma_i = \tau'_j$ and $M_i = M'_j$ already exists in its signature table. If so, it returns X'_j. Otherwise, it picks $\tau_i \neq \sigma_i$, computes y_i, μ_i and X_i in the same way as when answering a signature query, outputs X_i and adds $(i, y_i, \tau_i, \sigma_i, M_i)$ to its hash query table.

Assume F outputs a forgery (z, σ) on a message M. A then computes $X = F_\sigma(z^2)$ and searches its hash query table for such an i that $\sigma_i = \sigma$ and $M_i = M$. (Such an i has to exist for the same reasons as in the proof of Theorem 1.) Then, $F_\sigma(z^2) = X = \mu_i X_i = F_{\tau_i}(y_i^2)$, and $\sigma \neq \tau_i$ because of the way τ_i was picked, so A can easily factor n by the method given the proof of Lemma 1.

Probability Analysis. For this analysis, we assume that n is such that F has probability at least ε of success. By assumption, such n is generated with probability at least δ.

We need to compare the distribution of A's answers to F's queries to the distribution of answers given by the true oracles. The distributions differ as follows: if, for some signature query j, the hash value of τ'_j, M'_j has been already defined through a previous hash query, then A will fail, whereas the true oracles would not. The probability of this event for a given signature query is at most $(q_{hash}+1)2^{-k}$, because τ'_j is picked at random from $\{0,1\}^k$. Thus, the probability of F's success is reduced as a result of this by at most

$q_{sig}(q_{hash}+1)2^{-k}$. Additionally, X in the answers to hash and signature queries is distributed uniformly over $Z_n^* \cap \{0,1\}^{l-1}$, thus excluding the multiples of p_1 and p_2 that a real hash function could possibly hit. Since there are at most $p_1 + p_2 - 1 < 2^{l/2+1}$ such multiples in Z_n, this will reduce the forger's probability of success by at most $\frac{2^{l/2+1}}{2^{l-1}} = 2^{-l/2+2}$ per query.

So the probability of successfully factoring n is

$$\varepsilon' \geq \varepsilon - q_{sig}(q_{hash}+1)2^{-k} - (q_{hash}+q_{sig}+1)2^{-l/2+2}.$$

■

Theorem 4 *If there exists a forger that $(t, q_{sig}, q_{hash}, \varepsilon, \delta)$-breaks MSA-swap with security parameters l and k, such that $\varepsilon > \gamma$, then there exists an algorithm that $(t', \varepsilon', \delta)$-factors integers generated by Gen for*

$$\begin{aligned} t' &= \frac{t + 2(q_{sig}+q_{hash}+1)T_1}{\varepsilon - \gamma} + T_2 \\ \varepsilon' &= \left(1 - \frac{1}{e}\right) > 0.632, \end{aligned}$$

where T_1, T_2 and γ are as in Theorem 3.

Proof Let

$$\alpha = \varepsilon - \gamma.$$

By assumption, $\alpha > 0$. So if we repeat the algorithm constructed in the proof of Theorem 3 up to $1/\alpha$ times (except for the final gcd computation, which need only be done once), we will get the desired ε', similarly to the proof of Theorem 2. ■

Similarly to MSA, we have the following two immediate corollaries.

Corollary 3 *If factoring l-bit integers generated by Gen is $(t', \varepsilon', \delta)$-secure, then MSA-swap is $(t, q_{sig}, q_{hash}, \varepsilon, \delta)$-secure, where*

$$\begin{aligned} t &= t - 2(q_{sig}+q_{hash}+1)T_1 - T_2 \\ \varepsilon &= \varepsilon' + 2^{-l/2+2}(q_{hash}+q_{sig}+1) + 2^{-k}q_{sig}(q_{hash}+1). \end{aligned}$$

Proof The corollary follows simply from solving the equations of Theorem 3 for t and ε. ■

Corollary 4 *If factoring l-bit integers generated by Gen is $(t', 0.632, \delta)$-secure, then MSA-swap is $(t, q_{sig}, q_{hash}, \varepsilon, \delta)$-secure for*

$$t = \frac{(t'-T_2)\varepsilon}{2} - 2(q_{sig}+q_{hash}+1)T_1.$$

as long as

$$\varepsilon \geq 2\left(2^{-l/2+2}(q_{hash}+q_{sig}+1) + 2^{-k}q_{sig}(q_{hash}+1)\right)$$

Proof The condition on ε ensures that $\varepsilon - \gamma \geq \varepsilon/2$. The rest follows, similarly to the proof of Corollary 2, from solving the equations of Theorem 3 for t and ε. ■

4.4 Parameter Choice

The formulas in the Corollaries 1–4 are quite different. Nonetheless, it is immediately clear that *MSA-swap loses no factor of q_{hash},* neither in time nor in probability. This is a big advantage for MSA-swap because q_{hash} can be quite big.

A fuller comparison, provided in the next section, depends on the actual values of the parameters q_{sig}, q_{hash}, k and l. Let us deal here, however, with the preliminary problem of assigning reasonable values to these parameters.

We believe it reasonable to set $q_{sig} = 2^{30}$ and $q_{hash} = 2^{80} - 1$. This is so because signature queries have to be answered by the honest signer (who may not be willing or able to sign more than a billion messages), while hash queries can be computed by the adversary alone (who may be willing to invest extraordinary resources). Notice that we recommend a higher value for q_{hash} than suggested in [BR96].

We recommend setting $k = 100$ for MSA, and $k = 130$ for MSA-swap. For MSA, this is so because, from the Corollaries 3 and 4, we see that $2^{-k}(q_{hash} + 1)$ has to be small (the value of $2^{-k}(q_{hash} + 1)$ is essentially the success probability of the simple attack that relies on correctly guessing one hash value among $q_{hash} + 1$ hash queries). Therefore, we need 2^{-k+80} to be small, and by setting $k = 100$ we make it less than 10^{-6}. For MSA-swap, this is so because $2^{-k}q_{sig}(q_{hash} + 1)$ has to be small, as seen in the Corollaries 3 and 4.

As for l, notice that both MSA and MSA-swap are immediately broken if the adversary succeeds in factoring the l-bit modulus. Therefore, l ought to be *at least* 1000. Given the above choices for the other parameters, such a minimum value for l is large enough to make all the constraints involving l in Corollaries 1–4 satisfied (for any reasonable ε in the case of Corollaries 3 and 4). Thus, the value of l depends on the presumed security of factoring, as discussed in the next section.

5 Comparing Signature Schemes

Having developed the swap method and demonstrated that it tightens the security of Fiat-Shamir-like signature schemes, we wish to determine precisely when the method should be used. It would be tempting to say "whenever tight security is desired," but this answer is meaningless. We submit that meaningful comparisons f signature schemes should take both security and efficiency into account, and that consideration of either factor alone is insufficient.

We also use our methods to compare MSA to PRab from [BR96] and find, perhaps surprisingly, that the former may sometimes provide more security for less cost.

5.1 The Costs of Security

The desired level of security is usually dictated by the specific application. It is after settling on the desired amount of security that choosing among the various secure schemes becomes crucial. Indeed, when choosing a signature scheme, the goal is *to maintain the desired level of security at the lowest possible cost.* In a sense, picking a signature scheme is similar to shopping for an insurance policy for the desired face value.

The costs of a signature scheme, however, are quite varied. They may include the sizes of keys and signatures, the efficiencies of generating keys, signing and verifying, the amounts of code required, and even "external" considerations—such as the availability of inexpensive implementations or off-the-shelf hardware components. In this paper, we focus on the efficiencies of signing and verifying. These are particularly important when signing or verifying is performed by a low-power device, such as a smart card, or when signing or verifying needs to be performed in bulk quantities, as on a secure server.

It is for these costs, then, that below we compare MSA and MSA-swap. We also provide a comparison of MSA with the PRab scheme from [BR96], arguably the most practical among those *tightly* related to factoring. (The reason for choosing PRab rather than its PSS variant is that the latter is tightly related to RSA, and thus potentially less secure than factoring.)

5.2 Comparison of MSA and MSA-swap

The efficiency of signature verification in MSA is about the same as in MSA-swap. The security of MSA-swap is generally higher than the security of MSA for the same security parameters. Therefore, if the efficiency of verifying is the only significant component in the cost, MSA-swap will be able to provide the same amount of security for less cost than MSA.

A more difficult case to analyze is the case when the efficiency of signing is of main concern. We will limit our analysis to the case when we are only concerned with the on-line part of signing. In both cases, this involves mainly a modular exponentiation. Therefore, a variety of sophisticated algebraic methods can be used here, but these methods apply equally to MSA and MSA-swap. We thus find it simpler to compare the two under "standard" implementations using the Chinese remainder theorem (CRT). MSA then involves two exponentiations of a fixed $l/2$-bit base to a k-bit power, modulo two $l/2$-bit primes. One modular multiplication of two $l/2$-bit numbers takes about $l^2/4$ steps; about $3k$ such multiplications are required ($1.5k$ for each of the primes), so the total amount of time required for on-line signing in MSA is about $3kl^2/4$. Similar analysis applies for MSA-swap, except that about $1.5l$ multiplications ($1.5l/2$ for each of the two primes) are required, so the total amount of time required for on-line signing in MSA-swap is about $3l^3/8$, not counting the (relatively small) cost of computing the Jacobi symbol. (In sum, on-line signing is $l/(2k)$ times faster for MSA than for MSA-swap *if one used the same value of l for both.*[4])

Let us now see how the security of the two schemes compares assuming the on-line signing costs are the same. Let l_M and k_M be the security parameters for MSA, and l_{MS} and k_{MS} be the security parameters for MSA-swap. The on-line signing costs for MSA and MSA-swap are the same if

$$l_{MS} = (2k_M l_M^2)^{1/3}. \tag{1}$$

The best known factoring algorithms take time about

$$T(l) = C \exp\left(\left(\frac{64}{9}\right)^{1/3} l^{1/3} (\ln l)^{2/3}\right)$$

for some constant C [LL93]. Therefore, we will assume that factoring l-bit integers generated by *Gen* is $(C'T(l), 0.199, \delta)$-secure for some δ and some constant C'. This means that, for a large enough ε and small enough q_{sig}, MSA is about

$$\left(\frac{(C'T(l_M) - T_2(l_M, k_M))\varepsilon}{4q_{hash} + 6} - q_{sig} T_1(l_M, k_M), q_{sig}, q_{hash}, \varepsilon, \delta\right) - \text{secure}$$

and MSA-swap is about

$$\left(\frac{(C'T(l_{MS}) - T_2(l_{MS}, k_{MS}))\varepsilon}{2} - 2(q_{sig} + q_{hash} + 1) T_1(l_{MS}, k_{MS}), q_{sig}, q_{hash}, \varepsilon, \delta\right) - \text{secure}$$

(by Corollaries 2 and 4).

Keeping the signing costs equal, let us now find out when MSA becomes more secure than MSA-swap, that is, when

$$\frac{(C'T(l_M) - T_2(l_M, k_M))\varepsilon}{4q_{hash} + 6} - q_{sig} T_1(l_M, k_M) \quad > \quad \frac{(C'T(l_{MS}) - T_2(l_{MS}, k_{MS}))\varepsilon}{2} - 2(q_{sig} + q_{hash} + 1) T_1(l_{MS}, k_{MS})$$

[4]Moreover, an optimization available to MSA but not to MSA-swap is precomputing some powers of the fixed base; this requires additional memory, so we will assume it is not implemented for the purposes of this analysis.

Given the discussion of Section 4.4 and Equation (1), we will now plug in $q_{sig} = 2^{30}$, $q_{hash} = 2^{80} - 1$, $k_M = 100$, $k_{MS} = 130$, and $l_{MS} = (2k_M l_M^2)^{1/3}$ to the above inequality. These values allow us to omit the terms in T_1 and T_2, noting that they decrease the security of MSA-swap much more than they decrease the security of MSA. Thus, we are now interested in when

$$\frac{C'T(l_M)}{2^{82}+2} > \frac{C'T((200l_M^2)^{1/3})}{2}.$$

After taking the natural logarithm of both sides, we are left with the inequality

$$\left(\frac{64}{9}\right)^{1/3} l_M^{1/3}(\ln l_M)^{2/3} - \ln(2^{81}+1) > \left(\frac{64}{9}\right)^{1/3} (200l_M^2)^{1/9} \left(\frac{1}{3}\ln 200 + \frac{2}{3}\ln l_M\right)^{2/3}, \qquad (2)$$

which holds as long as long as $l_M \geq 6109$.

Thus, at $l_M = 6109$, $l_{MS} = 1954$, $k_M = 100$, $k_{MS} = 130$, MSA and MSA-swap provide about the security and the same performance for on-line signing.

In Sum. The on-line portion of the signing algorithm of MSA is so fast that

> *Provable security and signing efficiency are the same when MSA uses 6109-bit moduli and MSA-swap 1954-bit moduli.*

In both cases, the security is that of factoring a 1954-bit integer generated by *Gen*. (MSA may actually be even more secure, but we cannot prove it.)

It just so happens that this computed level of security is currently considered adequate for many applications. (Therefore, for these applications MSA-swap is preferable: MSA-swap has faster verification for the same level of security, as well as shorter keys and, therefore, shorter signatures.)

However, whenever the application calls for a *higher* level of security, and the dominant cost is that of signing, then it is the loosely-secure MSA that becomes preferable. In fact, Inequality (2) implies that the security gap between MSA and MSA-swap, given the same performance, increases exponentially. In essence, this is so because, in order to guarantee the same performance, l_{MS} is a fixed function of l_M: i.e., by Equation (1), $l_{MS} = O(l_M^{2/3})$. Thus security for both MSA and MSA-swap is measured by Inequality (2) in terms of l_M, and the dominant factor in the exponent of the adversary's running time is $l_M^{1/3}$ for MSA and $l_M^{2/9}$ for MSA-swap.

5.3 Comparison of MSA with Bellare-Rogaway's PRab

The security of PRab is tightly related to that of modular square roots, rather than factoring. A factor of 2 in probability is lost (as compared to MSA-swap) when one relates the security of PRab to that of factoring. PRab's performance for on-line signing is about the same as MSA-swap's (PRab requires a few more Jacobi symbol computations, but no separate modular multiplication).[5] A vastly similar analysis leads to the following conclusion:

> *Provable security and signing efficiency are the same when MSA uses 5989-bit moduli, and PRab 1929-bit moduli.*

Also here this is a "cross-over" point: the gap in security for the same performance increases exponentially in favor of MSA. As we can see, this cross-over point is just slightly more in favor of MSA than the cross-over point of MSA and MSA-swap. This is because of the factor of 2 difference in the security of MSA-swap and PRab.

[5]Unlike MSA-swap, however, PRab has no off-line component in signing. It also has very efficient verification.

Acknowledgments

We would like to thank Salil Vadhan for pointing out an error in an earlier version of this work and Mihir Bellare for suggesting using an idea from [BM99] to improve Corollary 1.

References

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

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

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In Maurer [Mau96], pages 399–416. Revised version appears in `http://www-cse.ucsd.edu/users/mihir/papers/crypto-papers.html`.

[FFS88] Uriel Feige, Amos Fiat, and Adi Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, 1988.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Odlyzko [Odl86], pages 186–194.

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

[Gol86] Oded Goldreich. Two remarks concerning the Goldwasser-Micali-Rivest signature scheme. In Odlyzko [Odl86], pages 104–110.

[Gol88] Shafi Goldwasser, editor. *Advances in Cryptology—CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990, 21–25 August 1988.

[GQ88] Louis Claude Guillou and Jean-Jacques Quisquater. A "paradoxical" indentity-based signature scheme resulting from zero-knowledge. In Goldwasser [Gol88], pages 216–231.

[LL93] Arjen K. Lenstra and Hendrik W. Lenstra, editors. *The development of the number field sieve*, volume 1554 of *Lecture notes in Mathematics*. Springer-Verlag, 1993.

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

[Mic94] Silvio Micali. A secure and efficient digital signature algorithm. Technical Report MIT/LCS/TM-501, Massachusetts Institute of Technology, Cambridge, MA, March 1994.

[MR99] Silvio Micali and Leonid Reyzin. Improving the exact security of Fiat-Shamir signature schemes. In R. Baumgart, editor, *Secure Networking — CQRE [Secure] '99*, volume 1740 of *Lecture Notes in Computer Science*, pages 167–182. Springer-Verlag, 1999.

[MS88] Silvio Micali and Adi Shamir. An improvement of the Fiat-Shamir identification and signature scheme. In Goldwasser [Gol88], pages 244–247.

[Odl86] Andrew M. Odlyzko, editor. *Advances in Cryptology—CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*. Springer-Verlag, 1987, 11–15 August 1986.

[Oka92] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In Ernest F. Brickell, editor, *Advances in Cryptology—CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 31–53. Springer-Verlag, 1993, 16–20 August 1992.

[OO88] Kazuo Ohta and Tatsuaki Okamoto. A modification of the Fiat-Shamir scheme. In Goldwasser [Gol88], pages 232–243.

[OO98] Kazuo Ohta and Tatsuaki Okamoto. On concrete security treatment of signatures derived from identification. In Hugo Krawczyk, editor, *Advances in Cryptology—CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 354–369. Springer-Verlag, 23–27 August 1998.

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

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

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000.

[Sch89] C. P. Schnorr. Efficient identification and signatures for smart cards. In J.-J. Quisquater and J. Vandewalle, editors, *Advances in Cryptology—EUROCRYPT 89*, volume 434 of *Lecture Notes in Computer Science*, pages 688–689. Springer-Verlag, 1990, 10–13 April 1989.

[Sch96] C. P. Schnorr. Security of 2^t-root identification and signatures. In Neal Koblitz, editor, *Advances in Cryptology—CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 143–156. Springer-Verlag, 18–22 August 1996.

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

[Wil80] Hugh C. Williams. A modification of the RSA public-key encryption procedure. *IEEE Transactions on Information Theory*, IT-26(6):726–729, November 1980.