A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks*

(Revision March 23, 1995)

Shafi Goldwasser**
Silvio Micali**
Ronald L. Rivest **

Abstract

We present a digital signature scheme based on the computational difficulty of integer factorization.

The scheme possesses the novel property of being robust against an adaptive chosen-message attack: an adversary who receives signatures for messages of his choice (where each message may be chosen in a way that depends on the signatures of previously chosen messages) can not later forge the signature of even a single additional message. This may be somewhat surprising, since the properties of having forgery being equivalent to factoring and being invulnerable to an adaptive chosen-message attack were considered in the folklore to be contradictory.

More generally, we show how to construct a signature scheme with such properties based on the existence of a "claw-free" pair of permutations – a potentially weaker assumption than the intractibility of integer factorization.

The new scheme is potentially practical: signing and verifying signatures are reasonably fast, and signatures are compact.

Keywords: Cryptography, digital signatures, factoring, chosen-message attacks, authentication, trapdoor permutations, randomization.

* This research was supported by NSF grants MCS-80-06938, DCR-8607494, and DCR-8413577, an IBM/MIT Faculty Development Award, and DARPA contract N00014-85-K-0125.

** MIT Laboratory for Computer Science, Cambridge, Mass. 02139

1. INTRODUCTION.

The idea of a "digital signature" first appeared in Diffie and Hellman's seminal paper, "New Directions in Cryptography"[DH76]. They propose that each user A publish a "public key" (used for validating signatures), while keeping secret a "secret key" (used for producing signatures). In their scheme user A's signature for a message M is a value which depends on M and on A's secret key, such that anyone can verify the validity of A's signature using A's public key. However, while knowing A's public key is sufficient to allow one to validate A's signatures, it does not allow one to easily forge A's signatures. They also proposed a way of implementing signatures based on "trap-door functions" (see section 2.1.1).

The notion of a digital signature is useful and is a legal replacement for handwritten signatures [LM78, Ma79]. However, a number of technical problems arise if digital signatures are implemented using trap-door functions as suggested by Diffie and Hellman [DH76]; these problems have been addressed and solved in part elsewhere. For example, [GMY83] showed how to handle arbitrary or sparse messages sets and how to ensure that if an enemy sees previous signatures (for messages that he has not chosen) it does not help him to forge new signatures (this is a "non-adaptive chosen-message attack" – see section 2.2).

The signature scheme presented here, using fundamentally different ideas than those presented by Diffie and Hellman, advances the state of the art of signature schemes with provable security properties even further; it has the following important characteristics:

- What we prove to be difficult is *forgery*, and not merely obtaining the secret key used by the signing algorithm (or obtaining an efficient equivalent algorithm).
- Forgery is proven to be difficult for a "most general" enemy who can mount an *adaptive chosen-message attack*. (An enemy who can use the real signer as "an oracle" can not in time polynomial in the size of the public key forge a signature for any message whose signature was not obtained from the real signer.) In contrast to all previous published work on this problem, we prove the scheme invulnerable against such an adaptive attack where each message whose signature is requested may depend on all signatures previously obtained from the real signer. We believe that an adaptive chosen-message attack is the most powerful attack possible for an enemy who is restricted to using the signature scheme in a natural manner.
- The properties we prove about the new signature scheme do not depend in any way on the set of messages which can be signed or on any assumptions about a probability distribution on the message set.
- Our scheme can be generalized so that it can be based on "hard" problems other than factoring whenever one can create claw-free trap-door pair generators.

Our scheme can be based on any family of pairs of claw-free permutations, yielding a signature scheme that is *invulnerable* to a chosen-message attack even if the claw-free trap-door permutations are *vulnerable* to a chosen-message attack when used to make a trap-door signature scheme (see section 2.1.1).

Fundamental ideas in the construction are the use of randomization, signing by using two authentication steps (the first step authenticates a random value which is used in the second step to authenticate the message), and the use of a tree-like branching authentication structure to produce short signatures.

We note that our signature scheme is not of the simple Diffie-Hellman "trap-door" type. For example, a given message can have many signatures.

Our signature scheme is seemingly "paradoxical", in that we prove that forgery is equivalent to factoring even if the enemy uses an *adaptive* chosen-message attack. We can restate the paradox as follows:

- Any general technique for forging signatures can be used as a "black box" in a construction that enables the enemy to factor one of the signer's public moduli (he has two in our scheme), but
- The technique of "forging" signatures by getting the real signer to play the role of the "black box" (i.e. getting the real signer to produce some desired genuine signatures) does not help the enemy to factor either of the signer's moduli.

Resolving this paradox was previously believed to be impossible and contradictory [Wi80, misled by Rivest].

The rest of this paper is organized as follows. In section 2 we present definitions of what it means to "break" a signature scheme and what it means to "attack" a signature scheme. In section 3 we review previously proposed signature schemes. In section 4 we review more closely the nature of the "paradox", and discuss how it canbe resolved. Section 5 defines some useful conventions and notation, and section 6 describes the complexity-theoretic foundations of our scheme. In section 7 we give the some of the fundamental notions for our signature scheme, and section 8 gives the details. In section 9 we prove that it has the desired properties. In the last section we discuss some ways to improve the running time and memory requirements of this scheme.

2. FUNDAMENTAL NOTIONS

To properly characterize the results of this paper, it is helpful to answer the following questions:

- What is a digital signature scheme?
- What kinds of attacks can the enemy mount against a digital signature scheme?
- What is meant by "breaking" the signature scheme?

Little attention has been so far devoted to precisely answer these questions. For instance, signature schemes have been generically called "secure" without specifying against what kind of attack. This way, it would not be surprising that "secure" signature schemes were later broken by an unforseen attack. We hope that the classification we propose in this section may prove useful in resolving unpleasant ambiguities.

2.1. What Is a Digital Signature Scheme?

A *digital signature scheme* contains the following components:

- A *security parameter* k, which is chosen by the user when he creates his public and secret keys. The parameter k determines a number of quantities (length of signatures, length of signable messages, running time of the signing algorithm, overall security, etc.).
- A *message space* $\mathcal{M}$ which is the set of messages to which the signature algorithm may be applied. Without loss of generality, we assume in this paper that all messages are represented as binary strings – that is $\mathcal{M} \subseteq \{0,1\}^+$. To ensure that the entire signing process is polynomial in the security parameter, we assume that the length of the messages to be signed is bounded by k^c, for some constant $c > 0$.
- A *signature bound* B which is an integer bounding the total number of signatures that can be produced with an instance of the signature scheme. This value is typically bounded above by a low-degree polynomial in k, but may be infinite.
- A *key generation algorithm* G which any user A can use on input 1^k (i.e. k in unary) to generate in polynomial time a pair (P_A^k, S_A^k) of matching *public* and *secret* keys. The secret key is sometimes called the *trap-door information*.
- A *signature algorithm* σ which produces a signature $\sigma(M, S_A)$ for a message M using the secret key S_A. Here σ may receive other inputs as well. For example, in the scheme we propose first, σ has an additional input which is the number of previously signed messages.
- A *verification algorithm* V which tests whether S is a valid signature for message M using the public key P_A. (I.e. $V(S, M, P_A)$ will be **true** if and only if it is valid.)

Any of the above algorithms may be "randomized" algorithms that make use of auxiliary random bit stream inputs. We note that G *must* be a randomized algorithm, since part of its output is the secret key, which must be unpredictable to an adversary. The signing algorithm σ may be randomized – we note in particular that our signing algorithm is randomized and is capable of producing many different signatures for the same message. In general, the verification algorithm need not be randomized, and ours is not.

We note that there are other kinds of "signature" problems which are not dealt with here; the most notable being the "contract signing problem" where two parties wish to exchange their signatures to an agreed-upon contract *simultaneously* (for example, see [Bl83], [EGL82], [BGMR85]).

2.1.1 A Classical Example: Trap-Door Signatures

To create a signature scheme, Diffie and Hellman proposed that A use a "trap-door function" f: informally, a function for which it is easy to evaluate $f(x)$ for any argument x but for which, given only $f(x)$, it is computationally infeasible to find *any* y with $f(y) = f(x)$ without the secret "trap-door" information. According to their suggestion, A publishes f and anyone can validate a signature by checking that $f(signature) = message$. Only A possesses the "trap-door" information allowing him to invert f:

$f^{-1}(message) = signature$. (Trap-door functions will be formally defined in section 6.) We call any signature scheme that fits into this model (i.e. uses trap-door functions and signs by apply f^{-1} to the message) a *trap-door signature scheme*.

We note that not all signature schemes are trap-door schemes, although most of the ones proposed in the literature are of this type.

2.2 Kinds of Attacks

We distinguish two basic kinds of attacks:

- **Key-Only Attacks** in which the enemy knows only the real signer's public key, and
- **Message Attacks** where the enemy is able to examine some signatures corresponding to either known or chosen-messages before his attempt to break the scheme.

We identify the following four kinds of message attacks, which are characterized by how the messages whose signatures the enemy sees are chosen. Here A denotes the user whose signature method is being attacked.

- **Known Message Attack:** The enemy is given access to signatures for a set of messages $m_1, \ldots, m_t$. The messages are known to the enemy but are not chosen by him.
- **Generic Chosen Message Attack:** Here the enemy is allowed to obtain from A valid signatures for a chosen list of messages $m_1, \ldots, m_t$ before he attempts to break A's signature scheme. These messages are *chosen* by the enemy, but they are *fixed* and *independent* of A's public key (for example the m_i's may be chosen at random). This attack is *nonadaptive*: the entire message list is constructed before any signatures are seen. This attack is "generic" since it does not depend on the A's public key; the same attack is used against everyone.
- **Directed Chosen Message Attack:** This is similar to the generic chosen-message attack, except that the list of messages to be signed may be created after seeing A's public key but before any signatures are seen. (The attack is still nonadaptive.) This attack is "directed" against a particular user A.
- **Adaptive Chosen Message Attack:** This is more general yet: here the enemy is also allowed to use A as an "oracle"; not only may he request from A signatures of messages which depend on A's public key but he may also request signatures of messages which depend additionally on previously obtained signatures.

The above attacks are listed in order of increasing severity, with the adaptive chosen-message attack being the most severe natural attack an enemy can mount. That the adaptive chosen-message attack is a natural one can be seen by considering the case of a notary public who must sign more-or-less arbitrary documents on demand. In general, the user of a signature scheme would like to feel that he may sign arbitrary documents prepared by others without fear of compromising his security.

2.3 What Does It Mean To "Break" a Signature Scheme?

One might say that the enemy has "broken" user A's signature scheme if his attack allows him to do any of the following with a non-negligible probability:

- **A Total Break:** Compute A's secret trap-door information.
- **Universal Forgery:** Find an efficient signing algorithm functionally equivalent to A's signing algorithm (based on possibly different but equivalent trap-door information).
- **Selective Forgery:** Forge a signature for a particular message chosen *a priori* by the enemy.
- **Existential Forgery:** Forge a signature for at least one message. The enemy has no control over the message whose signature he obtains, so it may be random or nonsensical. Consequently this forgery may only be a minor nuisance to A.

Note that to forge a signature means to produce a *new* signature; it is not forgery to obtain from A a valid signature for a message and then claim that he has now "forged" that signature, any more than passing around an authentic handwritten signature is an instance of forgery. For example, in a chosen-message attack it does not constitute selective forgery to obtain from the real signer a signature for the target message M.

Clearly, the kinds of "breaks" are listed above in order of decreasing severity – the least the enemy might hope for is to succeed with an existential forgery.

We say that a scheme is respectively *totally breakable*, *universally forgeable*, *selectively forgeable*, or *existentially forgeable* if it is breakable in one of the above senses. Note that it is more desirable to prove that a scheme is not even existentially forgeable than to prove that it is not totally breakable. The above list is not exhaustive; there may be other ways of "breaking" a signature scheme which fit in between those listed, or are somehow different in character.

We utilize here the most realistic notion of forgery, in which we say that a forgery algorithm succeeds if it succeeds probabilistically with a non-negligible probability. To make this notion precise, we say that the forgery algorithm succeeds if its chance of success is at least as large as one over a polynomial in the security parameter k.

To say that the scheme is "broken", we not only insist that the forgery algorithm succeed with a non-negligible probability, but also that it must run in probabilistic polynomial time.

We note here that the characteristics of the signature scheme may depend on its message space in subtle ways. For example, a scheme may be existentially forgeable for a message space $\mathcal{M}$ but not existentially forgeable if restricted to a message space which is a sufficiently small subset of $\mathcal{M}$.

The next section exemplifies these notions by reviewing previously proposed signature schemes.

3. PREVIOUS SIGNATURE SCHEMES AND THEIR SECURITY

In this section we list a number of previously proposed signature schemes and briefly review some facts about their security.

Trap-Door Signature Schemes [DH76]: Any trap-door signature scheme is existentially forgeable with a key-only attack since a valid (message, signature) pair can be created by beginning with a random "signature" and applying the public verification algorithm to obtain the corresponding "message". A common heuristic for handling this problem in practice is to require that the message space be sparse (i.e. requiring that very few strings actually represent messages – for example this can be enforced by having each message contain a reasonably long checksum.) In this case this specific attack is not likely to result in a successful existential forgery.

Rivest-Shamir-Adleman [RSA78]: The RSA scheme is selectively forgeable using a directed chosen-message attack, since RSA is *multiplicative*: the signature of a product is the product of the signatures. (This can be handled in practice as above using a sparse message space.)

Merkle-Hellman [MH78]: Shamir showed the basic Merkle-Hellman "knapsack" scheme to be universally forgeable using just a key-only attack [Sh82]. (This scheme was perhaps more an encryption scheme than a signature scheme, but had been proposed for use as a signature scheme as well.)

Rabin [Ra79]: Rabin's signature scheme is totally breakable if the enemy uses a directed chosen-message attack (see section 4). However, for non-sparse message spaces selective forgery is as hard as factoring if the enemy is restricted to a known message attack.

Williams [Wi80]: This scheme is similar to Rabin's. The proof that selective forgery is as hard as factoring is slightly stronger, since here only a single instance of selective forgery guarantees factoring (Rabin needed a probabilistic argument). Williams uses effectively (as we do) the properties of numbers which are the product of a prime $p \equiv 3 \pmod 8$ and a prime $q \equiv 7 \pmod 8$. Again, this scheme is totally breakable with a directed chosen-message attack.

Lieberherr [Li81]: This scheme is similar to Rabin's and Williams', and is totally breakable with a directed chosen-message attack.

Shamir [Sh78]: This knapsack-type signature scheme has recently been shown by Tulpan [Tu84] to be universally forgeable with a key-only attack for any practical values of the security parameter.

Goldwasser-Micali-Yao [GMY83]: This paper presents for the first time signature schemes which are not of the trap-door type, and which have the interesting property that their security characteristics hold for *any* message space. The first signature scheme presented in [GMY83] was proven not to be even existentially forgeable against a *generic* chosen-message attack unless factoring is easy. However, it is not known to what extent *directed* chosen-message attacks or adaptive chosen-message attacks might aid an enemy in "breaking" the scheme.

The second scheme presented there (based on the RSA function) was also proven not to be even existentially forgeable against a generic chosen-message attack. This scheme may also resist existentially forgery against an adaptive chosen-message attack, although this has not been proven. (A proof would require showing certain properties about the density of prime numbers and making a stronger intractability assumption about inverting RSA.) We might note that, by comparison, the scheme presented here is much faster, produces much more compact signatures, and is based on much simpler assumptions (only the difficulty of factoring or more generally the existence of claw-free permutation pair generators).

Several of the ideas and techniques presented in [GMY83], such as bit-by-bit authentication, are used in the present paper.

Ong-Schnorr-Shamir [OSS84a]: Totally breaking this scheme using an adaptive chosen-message attack has been shown to be as hard as factoring. However, Pollard [Po84] has recently been able to show that the "OSS" signature scheme is universally forgeable in practice using just a key-only attack; he developed an algorithm to forge a signature for any given message without obtaining the secret trap-door information. A more recent "cubic" version has recently been shown to be universally forgeable in practice using just a key-only attack (also by Pollard). An even more recent version [OSS84b] based on polynomial equations was similarly broken by Estes, Adleman, Kompella, McCurley and Miller [EAKMM85] for quadratic number fields.

El Gamal[EG84]: This scheme, based on the difficulty of computing discrete logarithms, is existentially forgeable with a generic message attack and selectively forgeable using a directed chosen-message attack.

Okamoto-Shiraishi[OS85]: This scheme, based on the difficulty of solving quadratic inequalities modulo a composite modulus, was shown to be universally forgeable by Brickell and DeLaurentis [BD85].

4. THE PARADOX OF PROVING SIGNATURE SCHEMES SECURE

The paradoxical nature of signature schemes which are provably secure against chosen-message attacks made its first appearance in Rabin's paper, "Digitalized Signatures as Intractable as Factorization" [Ra79]. The signature scheme proposed there works as follows. User A publishes a number n which is the product of two large primes. To sign a message M, A computes as M's signature one of M's square roots modulo n. (When M is not a square modulo n, A modifies a few bits of M to find a "nearby" square.) Here signing is essentially just extracting square roots modulo n. Using the fact that extracting square roots modulo n enables one to factor n, it follows that selective forgery in Rabin's scheme is equivalent to factoring if the enemy is restricted to at most a known message attack.

However, it is true (and was noticed by Rabin) that an enemy might totally break the scheme using a directed chosen-message attack. By asking A to sign a value $x^2 \bmod n$ where x was picked at random, the enemy would obtain with probability $\frac{1}{2}$ another square root y of x^2 such that $\gcd(x+y, n)$ was a prime factor of n.

Rabin suggested that one could overcome this problem by, for example, having the signer concatenate a fairly long randomly chosen pad U to the message before signing it. In this way the enemy can not force A to extract a square root of any particular number.

However, the reader may now observe that the proof of the equivalence of selective forgery to factoring no longer works for the modified scheme. That is, being able to selectively forge no longer enables the enemy to directly extract square roots and thus to factor. Of course, breaking this equivalence was really the whole point of making the modification.

4.1 The Paradox

We now "prove" that it is impossible to have a signature scheme for which it is both true that forgery is provably equivalent to factoring, and yet the scheme is invulnerable to adaptive chosen-message attacks. The argument is essentially the same as the one given in [Wi80]. By *forgery* we mean in this section any of universal, selective, or existential forgery – we assume that we are given a proof that forgery of the specified type is equivalent to factoring.

Let us begin by considering this given proof. The main part of the proof presumably goes as follows: given a subroutine for forging signatures, a constructive method is specified for factoring. (The other part of the equivalance, showing that factoring enables forgery, is usually easy, since factoring usually enables the enemy to totally break the scheme.)

But it is trivial then to show that an adaptive chosen-message attack enables an enemy to totally break the scheme. The enemy merely executes the constructive method for factoring given in the proof, using the real signer instead of the forgery subroutine! That is, whenever he needs to execute the forgery subroutine to obtain the signature of a message, he merely performs an "adaptive chosen-message attack" step – getting the real user to sign the desired message. In the end the unwary user has enabled the enemy to factor his modulus! (If the proof reduces factoring to universal or selective forgery, the enemy has to get the real user to sign a particular message. If the proof reduces factoring to existential forgery, the enemy need only get him to sign anything at all.)

4.2 Breaking The Paradox

How can one hope to get around the apparent contradictory natures of equivalence to factoring and invulnerability to an adaptive chosen-message attack?

The key idea in resolving the paradox is to have the constructive proof that forgery is as hard as factoring be a *uniform* proof which makes *essential* use of the fact that the forger can forge for *arbitrary* public keys with a non-negligible probability of success. However, in "real life" a signer will only produce signatures for a *particular* public key. Thus the constructive proof can not be applied in "real life" (by asking the real signer to unwittingly play the role of the forger) to factor.

In our scheme this concept is implemented using the notion of "random rooting". Each user publishes not only his two composite moduli n_1 and n_2, but also a "random root" r. This value r is used when validating the user's signatures. The paradox is resolved in our case as follows:

- It is provably equivalent to factoring for an enemy to have a *uniform* algorithm for forging; uniform in the sense that if for all pairs of composite numbers n_1 and n_2 if the enemy can randomly forge signatures for a significant fraction of the possible random roots r, then he can factor either n_1 or n_2.
- The above proof *requires* that the enemy be able to pick r himself – the forgery subroutine is fed triples (n_1, n_2, r) where the r part is chosen by the enemy according the procedure specified in the constructive proof. However, in "real life" the user has picked a *fixed* r at random to put in his public key, so an adaptive chosen-message attack will not enable the enemy to "forge" signatures corresponding to any other values of r. Thus the constructive method given in the proof can not be applied! More details can be found in section 9.

5. GENERAL NOTATION AND CONVENTIONS

5.1 Notation and Conventions for Strings

Let $\alpha = \alpha_0\alpha_1 \ldots \alpha_x$ be a binary string, then $\bar{\alpha}$ will denote the integer $\sum_{k=0}^{x} \alpha_k 2^{x-k}$. (Note that a given integer may have several denotations, but only one of a given length.) The strings in $\{0,1\}^*$ are ordered as follows: if α and β are binary strings, we write $\alpha < \beta$ if there exists a string γ such that α is a prefix of γ, γ has exactly the same length as β, and $\bar{\gamma} < \bar{\beta}$.

If i is a k-bit string, we let $DFS(i) = \{\beta \mid \beta \leq i\}$. (Imagine a full binary tree of depth k whose root is labelled ϵ, and the left (right) son of a node labelled α is $\alpha 0$ ($\alpha 1$) and let DFS be the Depth First Search algorithm that starts at the root and explores the left son of any node before the right son of that node. Then $DFS(i)$ represents the set of nodes visited by DFS up to and including the time when it reaches node i). Note that $DFS(i)$ contains the empty string.

5.2 Notation and Conventions for Probabilistic Algorithms.

We introduce some generally useful notation and conventions for discussing probabilistic algorithms. (We make the natural assumption that all parties, including the enemy, may make use of probabilistic methods.)

We emphasize the number of inputs received by an algorithm as follows. If algorithm A receives only one input we write "$A(\cdot)$", if it receives two inputs we write "$A(\cdot,\cdot)$" and so on.

We write "PS" for "probability space"; in this paper we only consider countable probability spaces. In fact, we only deal with probability spaces arising from probabilistic algorithms.

If $A(\cdot)$ is a probabilistic algorithm then, for any input i, the notation $A(i)$ refers to the PS which assigns to the string σ the probability that A, on input i, outputs σ. We point out the special case that A takes no inputs; in this case the notation A refers to the algorithm itself, whereas the notation $A()$ refers to the PS defined by running A with no input. If S is a PS, we denote by $\mathbf{P}_S(e)$ the probability that S associates with element e. Also, we denote by $[S]$ the set of elements which S gives positive probability. In the case that $[S]$ is a singleton set $\{e\}$ we will use S to denote the value e; this is in agreement with traditional notation. (For instance, if $A(\cdot)$ is an algorithm that, on input i, outputs i^3, then we may write $A(2) = 8$ instead of $[A(2)] = \{8\}$.)

If $f(\cdot)$ and $g(\cdot, \cdots)$ are probabilistic algorithms then $f(g(\cdot, \cdots))$ is the probabilistic algorithm obtained by composing f and g (i.e. running f on g's output). For any inputs $x, y, \ldots$ the associated probability space is denoted $f(g(x, y, \ldots))$.

If S is a PS, then $x \leftarrow S$ denotes the algorithm which assigns to x an element randomly selected according to S; that is, x is assigned the value e with probability $\mathbf{P}_S(e)$.

The notation $\mathbf{P}(p(x, y, \ldots) | x \leftarrow S; y \leftarrow T; \ldots)$ will then denote the probability that the predicate $p(x, y, \ldots)$ will be true, after the (ordered) execution of the algorithms $x \leftarrow S$, $y \leftarrow T$, etc.

We let $\mathcal{RA}$ denote the set of probabilistic polynomial-time algorithms. We assume that a natural representation of these algorithms as binary strings is used.

By 1^k we denote the unary representation of integer k, i.e.

$$\underbrace{11 \ldots 1}_{k}$$

6. THE COMPLEXITY THEORETIC BASIS OF THE NEW SCHEME

A particular instance of our scheme can be constructed if integer factorization is computationally difficult. However, we will present our scheme in a general manner without assuming any particular problem to be intractable. This clarifies the exposition, and helps to establish the true generality of the proposed scheme. We do this by introducing the notion of a "claw-free permutation pair", and constructively showing the existence of such objects under the assumption that integer factorization is difficult.

This section builds up the relevant concepts and definitions in stages. In subsection 6.1. we give a careful definition of the notions of a trap-door permutation and a trap-door permutation generator. These notions are not directly used in this paper, but serve as a simple example of the use of our notation. (Furthermore, no previous definition in the literature was quite so comprehensive.) The reader may, if he wishes, skip section 6.1 without great loss.

In subsection 6.2. we define claw-free permutation pairs and claw-free permutation pair generators.

In subsection 6.3. we show how to construct claw-free permutation pair generators under the assumption that factoring is difficult.

Finally, in subsection 6.4. we show how to construct an infinite family of pairwise claw-free permutations, given a generating pair f_0, f_1, of claw-free permutations.

Altogether, then, this section provides the underlying definitions and assumptions required for constructing our signature scheme. The actual construction of our signature scheme will be given in sections 7 and 8.

6.1 Trap-door Permutations

Informally, a family of trap-door permutations is a family of permutations f possessing the following properties:

- It is easy, given a integer k, to randomly select permutations f in the family which have k as their security parameter, together with some extra "trap-door" information allowing easy inversion of the permutations chosen.
- It is hard to invert f without knowing f's trap-door.

We can interpret the two properties above by saying that any user A can easily randomly select a pair of permutations, (f, f^{-1}), inverses of each other. This will enable A to easily evaluate and invert f; if now A publicizes f and keeps secret f^{-1}, then inverting f will be hard for all other users.

In the informal discussion above, we used the terms "easy" and "hard". The term "easy" can be interpreted as "in polynomial time"; "hard", however, is of more difficult interpretation. By saying that f is hard to invert we cannot possibly mean that f^{-1} cannot be easily evaluated at any of its arguments.* We mean, instead, that f^{-1} is hard to evaluate at a *random* argument. Thus, if one wants (as we do) to use trap-door functions to generate problems computationally hard for an "adversary", he must be able to randomly select a point in the domain of f and f^{-1}. This operation is easy for all currently known candidates of a trap-door permutation, and we explicitly assume it to be easy in our formal treatment.

Definition: Let G be an algorithm in $\mathcal{RA}$ that on input 1^k, outputs an ordered triple (d, f, f^{-1}) of algorithms. (Here $D = [d()]$ will denote the domain of the trap-door permutation f and its inverse f^{-1}.) We say that G is a *trap-door permutation generator* if there is a polynomial p such that

(1) Algorithm d always halts within $p(k)$ steps and defines a uniform probability distribution over the finite set $D = [d()]$. (I.e., running d with no inputs uniformly selects an element from D.)

(2) Algorithms f and f^{-1} halt within $p(k)$ steps on any input $x \in D$. (For inputs x not in D, the algorithms f and f^{-1} either loop forever or halt and print an error message that the input is not in the appropriate domain.) Furthermore, the functions $x \mapsto f(x)$ and $x \mapsto f^{-1}(x)$ are inverse permutations of D.

(3) For all (inverting) algorithms $I(\cdot,\cdot,\cdot,\cdot) \in \mathcal{RA}$, for all c and sufficiently large k:

$$\mathbf{P}(y = f^{-1}(z) | (d, f, f^{-1}) \leftarrow G(1^k); z \leftarrow d(); y \leftarrow I(1^k, d, f, z)) < k^{-c}.$$

We make the following informal remarks corresponding to parts of the above definition.

(1) This condition makes it explicit that it is possible to sample the domain of f in a uniform manner.

(3) This part of the definition states that if we run the experiment of generating (d, f, f^{-1}) using the generator G and security parameter k, and then randomly generating an element z in the range of f, and then running the "inverting" algorithm I (for polynomially in k many steps) on inputs d, f, and z, the chance that I will successfully invert f at the point z is vanishingly small as a function of k.

Definition: If G is a trap-door permutation generator, we say that $[G(1^k)]$ is a *family of trap-door permutations*. We say that f and f^{-1} are *trap-door permutations* if $(d, f, f^{-1}) \in [G(1^k)]$ for some k and trap-door permutation generator G.

6.2 "Claw-Free" Permutation Pairs

The signature scheme we propose is based on the existence of "claw-free" permutation pairs – informally, these are permutations f_0 and f_1 over a common domain for which it is computationally infeasible to find a triple x, y, and z such that $f_0(x) = f_1(y) = z$ (a "claw" or "f-claw" – see Figure 1).

Figure 1. A Claw

* For example, any f can be easily inverted at the image of a fixed argument, say 0. In fact, we may consider inverting algorithms that, on inputs x and f, first check whether $x = f(0)$.

Definition: Let G be an algorithm in $\mathcal{RA}$ that, on input 1^k, outputs an ordered quintuple $(d, f_0, f_0^{-1}, f_1, f_1^{-1})$ of algorithms. We say that G is a *claw-free permutation pair generator* if there is a polynomial p such that:

(1) Algorithm d always halts within $p(k)$ steps and defines a uniform probability distribution over the finite set $D = [d()]$.

(2) Algorithms f_0, f_0^{-1}, f_1 and f_1^{-1} halt within $p(k)$ steps on any input $x \in D$. (For inputs x not in D, these algorithms either loop forever or halt with an error message that the input is not in the necessary domain.) Furthermore, the functions $x \mapsto f_0(x)$ and $x \mapsto f_0^{-1}(x)$ are permutations of D which are inverses of each other, as are $x \mapsto f_1(x)$ and $x \mapsto f_1^{-1}(x)$.

(3) For all (claw-making) algorithms $I(\cdot,\cdot,\cdot,\cdot) \in \mathcal{RA}$, for all c and sufficiently large k:

$$\mathbf{P}(f_0(x) = f_1(y) = z | (d, f_0, f_0^{-1}, f_1, f_1^{-1}) \leftarrow G(1^k); (x, y, z) \leftarrow I(1^k, d, f_0, f_1)) < k^{-c}.$$

Note: It would be possible to use a variant of the above definition, in which the function f may actually return answers for inputs outside of D, as long as it is understood that the difficulty of creating a "claw" applies to all x, y for which the function f returns an answer. Thus, it should be hard to find *any* triplet (x, y, z) such that $f_0(x) = f_1(y) = z$ even when x, y are not in D. We do not pursue this variation further in this paper.

Definition: We say that $f = (d, f_0, f_1)$ is a *claw-free permutation pair* (or *claw-free pair* for short) if $(d, f_0, f_0^{-1}, f_1, f_1^{-1}) \in [G(1^k)]$ for some k and claw-free permutation pair generator G. In this case, f^{-1} will denote the pair of permutations (f_0^{-1}, f_1^{-1}).

6.2.1 Claw-Free Permutation Pairs vs. Trapdoor Permutations

In this subsection we clarify the relation between the notions of claw-free permutation pairs and trapdoor permutations, by showing that the existence of the former ones implies the existance of the latter ones. (Since trapdoor permutations are not used in our signature scheme, this subsection can be skipped by the reader without loss of clarity.)

Claim: Let $G \in \mathcal{RA}$ be a claw-free permutation generator. Then there exists a $\bar{G} \in \mathcal{RA}$ which is a trapdoor permutation generator.

Proof: The algorithm $\bar{G}$ is defined as follows on input 1^k: Run G on input 1^k. Say, G outputs the ordered tuple $(d, f_0, f_0^{-1}, f_1, f_1^{-1})$. Then, $\bar{G}$ outputs (d, f_0, f_0^{-1}).

We now show that $\bar{G}$ is a trapdoor permutation generator. Assume for contradiction that it not the case. Namely, there exists a constant $c > 0$ and an inverting algorithm $\bar{I}(\cdot,\cdot,\cdot,\cdot) \in \mathcal{RA}$ such that for infinitely many k:

$$\mathbf{P}(f_0(y) = z | (d, f_0, f_0^{-1}) \leftarrow \bar{G}(1^k); z \leftarrow d(); y \leftarrow \bar{I}(1^k, d, f_0, z)) \geq k^{-c}.$$

Note now, that since f_1 is a permutation, algorithms $f_1(d(\cdot))$ and $d(\cdot)$ both define the uniform probability distribution over $[d()]$. Thus, for infinitely many k,

$$\mathbf{P}(f_1(x) = f_0(y) = z | (d, f_0, f_0^{-1}, f_1, f_1^{-1}) \leftarrow G(1^k); x \leftarrow d(); z \leftarrow f_1(x); y \leftarrow \bar{I}(1^k, d, f_0, z)) \geq k^{-c}.$$

Let $I(\cdot,\cdot,\cdot,\cdot)$ be the following inverting algorithm: On input $1^k, d, f_0$, and f_1, compute $x \leftarrow d()$, $z \leftarrow f_1(x)$, $y \leftarrow \bar{I}(1^k, d, f_0, z)$, and output (x, y, z).

Then, I is in $\mathcal{RA}$ and for infinitely many k,

$$\mathbf{P}(f_0(x) = f_1(y) = z | (d, f_0, f_0^{-1}, f_1, f_1^{-1}) \leftarrow G(1^k); (x, y, z) \leftarrow I(1^k, d, f_0, f_1)) > k^{-c}.$$

This contradicts G being a claw-free permutation generator and thus $\bar{G}$ must be a trapdoor permutation generator.

∎

We note, however, that the the converse to the above claim may be false. For example, the pair of ("RSA") permutations over $Z_n^* = \{1 \leq x \leq n : gcd(x, n) = 1\}$, defined by

$$f_0(x) \equiv x^3 \pmod{n}, \text{ and}$$

$$f_1(x) \equiv x^5 \pmod{n}$$

(where $\gcd(\phi(n), 15) = 1$) is not claw-free : since the two functions commute it is easy to create a claw by choosing w at random and then defining $x \equiv f_1(w)$, $y \equiv f_0(w)$, and

$$z \equiv f_0(x) \equiv f_1(y) \equiv w^{15} \pmod{n}.$$

However, it is likely that f_0 and f_1 are trap-door permutations.

In practice, one may want to relax the definition of a claw-free permutation pair generator slightly, to allow the generator to have a very small chance of outputting functions f_0 and f_1 which are not permutations. We do not pursue this line of development in this paper.

6.3 Claw-free permutations exist if factoring is hard

The assumption of the existence of claw-free pairs is made in this paper in a general manner, independent of any particular number theoretic assumptions. Thus instances of our scheme may be secure even if factoring integers turns out to be easy. However for concretely implementing our scheme the following is suggested.

We first make an assumption about the intractability of factoring, and then exhibit a claw-free permutation pair generator based on the difficulty of factoring.

Notation: Let

$$H_k = \{n = p \cdot q \mid |p| = |q| = k, p \equiv 3 \pmod{8}, q \equiv 7 \pmod{8}\}$$

(the set of composite numbers which are the product of two k-bit primes which are both congruent to 3 modulo 4 but not congruent to each other modulo 8), and let $H = \bigcup_k H_k$.

Remark: One way to choose "hard" instances for all known factoring algorithms seems to be to choose k to be large enough and then to choose n randomly from H_k.

These numbers were used in [Wi80] and their wide applicabilty to cryptography was demonstrated by Blum in [Bl82] – hence they are commonly referred to as "Blum integers".

Let Q_n denote the set of quadratic residues (mod n). We note that for $n \in H$:

-1 has Jacobi symbol $+1$ but is not in Q_n.

2 has Jacobi symbol -1 (and is not in Q_n).

We also note every $x \in Q_n$ has exactly one square root $y \in Q_n$, but has four square roots $y, -y, w, -w$ altogether (see [Bl82] for proof). Roots w and $-w$ have Jacobi symbol -1, while y and $-y$ have Jacobi symbol $+1$.

The following assumption about the intractability of factoring is made throughout this subsection.

Intractability Assumption for Factoring (IAF): Let A be a probabilistic polynomial-time (factoring) algorithm. Then for all constants $c > 0$ and sufficiently large k

$$\mathbf{P}(x \quad \text{is a nontrivial divisor of } n | n \leftarrow H_k(); x \leftarrow A(n)) < \frac{1}{k^c}.$$

(Here we have used the notation $n \leftarrow H_k()$ to denote the operation of selecting an element of H_k uniformly at random.)

Define $f_{0,n}$ and $f_{1,n}$ as follows:

$$f_{0,n}(x) = \begin{cases} x^2 \pmod{n} & \text{if } x^2 \pmod{n} < n/2; \\ -x^2 \pmod{n} & \text{if } x^2 \pmod{n} > n/2. \end{cases}$$

$$f_{1,n}(x) = \begin{cases} 4x^2 \pmod{n} & \text{if } 4x^2 \pmod{n} < n/2; \\ -4x^2 \pmod{n} & \text{if } 4x^2 \pmod{n} > n/2. \end{cases}$$

The common domain of these functions is

$$D_n = \{x \in Z_n \mid \left(\frac{x}{n}\right) = 1 \quad \& \quad 0 < x < n/2\};$$

it is easy to see that the range of these functions is included in D_n for $n \in H$. Note also that it is easy to test whether or not a given element x is a member of D_n, since Jacobi symbols can be evaluated in polynomial time.

We now show that $f_{0,n}$ and $f_{1,n}$ are actually permutations of D_n for $n \in H$. Suppose $f_{0,n}$ is not a permutation of D_n; then there exist distinct elements x, y in D_n such that $f_{0,n}(x) = f_{0,n}(y)$. This can only happen if $x^2 \equiv y^2 \pmod{n}$, which would imply that $x \equiv \pm y \pmod{n}$. But this is impossible if x and y are both in D_n, thus proving that $f_{0,n}$ is a permutation. The proof for $f_{1,n}$ is similar.

Not only are $f_{0,n}$ and $f_{1,n}$ permutations of D_n when $n \in H$, but their inverses are easily computed, *given knowledge of p and q*. Given p and q, it is easy to distinguish quadratic residues (mod n) from residues with Jacobi symbol equal to 1; this ability enables one to negate the input to the inverse function if necessary in order to obtain a quadratic residue (mod n). Of course, dividing by 4 is easy – this step is needed only for inverting $f_{1,n}$. Next, taking square roots (mod n) is easy, since we can take square roots modulo p and q separately (making sure to pick the square root which is itself a quadratic residue) and combine the results using the Chinese Remainder Theorem. Finally, the result can be negated (mod n) as necessary in order to obtain a result in D_n. Since all of these steps are computable in polynomial time, each of the inverse functions $f_{0,n}^{-1}$ and $f_{1,n}^{-1}$ is computable in polynomial time, given p and q as additional inputs.

Theorem 1: Under the IAF, the following algorithm G is a claw-free permutation pair generator. On input 1^k, G:

(1) Generates two random primes p and q of length k, where $p \equiv 3 \pmod{8}$ and $q \equiv 7 \pmod{8}$.

(2) Outputs the quintuple

$$(d, f_{0,n}, f_{0,n}^{-1}, f_{1,n}, f_{1,n}^{-1})$$

where

(a) Algorithm d generates elements uniformly at random in Q_n.

(b) Algorithms $f_{0,n}$ and $f_{1,n}$ are as described in the above equations.

(c) Algorithms $f_{0,n}^{-1}$ and $f_{1,n}^{-1}$ are algorithms for the inverse functions (these algorithms make use of p and q).

Proof: We first note that uniformly selecting k-bit guaranteed primes can be accomplished in expected polynomial (in k) time, by the recent work of Goldwasser and Kilian [GK86], and that asymptotically one-quarter of these will be congruent to $3 \pmod{8}$ (similarly for those congruent to $7 \pmod{8}$). (In practice, one would use a faster probabilistic primality test such as the one proposed by Solovay and Strassen [SS77] or Rabin [Ra80].)

Let $n \in H$ and $(d, f_{0,n}, f_{0,n}^{-1}, f_{1,n}, f_{1,n}^{-1}) \in [G(1^k)]$. First, $f_{0,n}$ and $f_{1,n}$ are permutations of $D_n = [d()]$. Then, we need only show that if there exists a fast algorithm that finds x and y in D_n such that $f_{0,n}(x) \equiv f_{1,n}(y) \pmod{n}$ (i.e. a claw-creating algorithm) then factoring is easy. Suppose such an x and y have been found. Then $x^2 \equiv 4y^2 \pmod{n}$. (Note that $x^2 \equiv -4y^2 \pmod{n}$ is impossible: since $4y^2$ is a quadratic residue (mod n), $-4y^2$ can not be a quadratic residue (mod n), for $n \in H$.) This implies that $(x+2y)(x-2y) \equiv 0 \pmod{n}$. Moreover, we also know that $x \not\equiv \pm 2y \pmod{n}$, since $\left(\frac{x}{n}\right) = 1$ and $\left(\frac{2y}{n}\right) = -1$. Thus $\gcd(x \pm 2y, n)$ will produce a nontrivial factor of n. ∎

6.4 An Infinite Set of Pairwise Claw-Free Permutations

For our scheme we need not just claw-free pairs of permutations, but an infinite family of permutations which are pairwise claw-free and generated by a single claw-free pair $f = (d, f_0, f_1)$.

We define the function $f_i(\cdot)$ for any string $i \in \{0,1\}^+$ by the equation:

$$f_i(x) = f_{i_0}(f_{i_1}(\ldots(f_{i_{d-1}}(f_{i_d}(x))\ldots)))$$

if $i = i_0 i_1 \ldots i_{d-1} i_d$. (Also, read f_i^{-1} as $(f_i)^{-1}$ so that $f_i^{-1}(f_i(x)) = x$.)

Each f_i is a trap-door permutation: it is easy to compute $f_i(x)$ given f_0, f_1, i, and x, and to compute $f_i^{-1}(x)$ if f_0^{-1} and f_1^{-1} are available. However, given only f_0 and f_1 it should be hard to invert f_i on a random input z, or else f_0 and f_1 are not trap-door permutations. (By inverting f_i on a random input one also effectively inverts f_{i_0} on a random input, where i_0 is the first bit of i.)

This way of generating an infinite family of trap-door permutations was also used in [GMY83].

Looking ahead, we shall see that a user A of our scheme can use the f_i's to perform basic authentication steps as follows. Let us presume that A has published f_0 and f_1 as part of his public key, and has kept their inverses f_0^{-1} and f_1^{-1} secret. If user A is known to have authenticated a string y, then by publishing strings i and x such that

$$f_i(x) = y,$$

he thereby authenticates the new strings i and x.

For this to work, when the signer A reveals $f_i^{-1}(y)$ he should not enable anyone else to compute $f_j^{-1}(y)$ for any other j.

The signer achieves this in our scheme by coding i using a prefix-free mapping $\langle\cdot\rangle$. This prevents an enemy from computing $f_{\langle j\rangle}^{-1}(x)$ from $f_{\langle i\rangle}^{-1}(x)$ in an obvious way since $\langle j\rangle$ is never a prefix of $\langle i\rangle$. The following lemma 1 shows that this approach is not only necessary but sufficient.

Note: Actually, the mapping $\langle\cdot\rangle$ that we use is a one-to-one mapping from *tuples* of strings of bits to strings of bits. The mapping $\langle\cdot\rangle$ is prefix-free in the sense that $\langle a_1, \ldots, a_n\rangle$ is never a prefix of $\langle b_1, \ldots, b_m\rangle$ unless $n = m$ and $a_1 = b_1, \ldots, a_n = b_n$. Any prefix-free mapping is usable if it and its (partial) inverses are polynomial-time computable and the lengths of $a_1, \ldots, a_n$ and $\langle a_1, \ldots, a_n\rangle$ are polynomially related. For concreteness, we suggest the following encoding scheme for the tuple of strings $a_1, \ldots, a_n$. Each string a_i is encoded by changing each 0 to 00 and each 1 to 11, and the encoding is followed by 01. The encodings of $a_1, \ldots, a_n$ are concatenated and followed by 10.

Lemma 1 essentially says that if (d, f_0, f_1) is a claw-free pair, then it will be hard to find two different tuples of strings i and j, and elements x and y such that $f_{<i>}(x) = f_{<j>}(y)$.

Lemma 1: Let $f = (d, f_0, f_1)$ be a claw-free pair, x and y be elements of d and i, j two different tuples of binary strings such that there exists a string z such that $z = f_{\langle i\rangle}(x) = f_{\langle j\rangle}(y)$. Then there exists an f-claw (x_1, x_2, x_3) where $x_3 = f_c^{-1}(z)$ for some prefix c of $\langle i\rangle$.

Proof: Let $c \in \{0,1\}^*$ be the longest common prefix of $\langle i\rangle$ and $\langle j\rangle$. Such a c must exist since $\langle\cdot\rangle$ is a prefix-free encoding scheme. Thus, setting $x_3 \leftarrow f_c^{-1}(z)$, $x_1 \leftarrow f_{c0}^{-1}(z)$, and $x_2 \leftarrow f_{c1}^{-1}(z)$, we obtain an f-claw (x_1, x_2, x_3). (If c is the empty string then f_c^{-1} denotes the identity function, so $x_3 = z$.) Note that the f-claw is easily computed from f, x, and y. ∎

7. BUILDING BLOCKS FOR SIGNING

In this section we define the basic building blocks needed for describing our signature scheme. In section 8, we will define what a signature is and how to sign, using the objects and data structures introduced here.

Assumption: We assume from here on that all claw-free functions used are defined over domains which do *not* include the empty string ϵ.

This assumption is necessary since we use ϵ as a "marker" in our construction; note that it is easy, via simple recodings, to enforce this construction if necessary.

We begin by defining the essential notion of an f-item.

Definition: Let $f = (d_f, f_0, f_1)$ be a claw-free pair. A tuple of strings $(t, r; c_1, \ldots, c_m)$ is an *f-item* if

$$f_{\langle c_1,\ldots,c_m\rangle}(t) = r$$

Definition: In an f-item $(t, r; c_1, \ldots, c_m)$,

- t is called the *tag* of the item,
- r is called the *root* of the item, and
- the c_i's are the *children* of the item. We note that the children are ordered, so that we can speak of the first child or the second child of the item.

Note that given a claw free pair f and a tuple it is easy to check if the tuple is an f-item by applying the appropriate $f_{\langle i\rangle}$ to the tag, and checking if the correct root is obtained.

Figure 2 gives our graphic representation of an f-item $(t, r; c_1, c_2)$ with two children.

Figure 2. An f-item with two children

Definition: We say that a sequence of f-items $L_1, L_2, \ldots, L_b$ is an *f-chain starting at y* if, for $i = 1, \ldots, b-1$, the root of L_{i+1} is one of the children of L_i and y is the root of L_1. We say the chain *ends at x* if x is one of the children of the item L_b.

For efficiency considerations, our signature scheme will organize a collection of a special type of f-chains in the tree-like structure defined below.

Definition: Let i be a binary string of length b and f a claw-free pair. An *f-i-tree* is a bijection T between $DFS(i)$ and a set of f-items such that:

(1) if string j has length b, then $T(j)$ is an f-item with exactly two children, exactly one of which is ϵ, the empty string. These f-items are called *bridge items.*

(2) if string j has length less than b, then $T(j)$ is an f-item with exactly two children, c_0 and c_1, both of which are non-empty strings. Moreover, c_0, the *0th child*, is the root of $T(j0)$ and c_1, the *1st child*, the root of $T(j1)$.

The f-item $T(j)$ is said to be of *depth d* if string j has length d. (The bridge items are thus the items of depth b.) The *root* of T is the root of the f-item $T(\epsilon)$. The *internal nodes* of T are the root and the children of the f-items of depth less than b. The *leaves* of T are the non-empty children of the bridge items. Thus the internal nodes and the leaves of an f-i-tree are actual values and not f-items. Leaves possess binary names of length b, leaf j is the non-empty child of bridge item $T(j)$. The *path to leaf* $j = j_0 \ldots j_b$ is the f-chain $T(\epsilon), T(j_0), \ldots, T(j_0 \ldots j_b)$.

Figure 3 gives our graphic representation of an f-100-tree, as it would be used in our signature scheme. In this figure we denote by r_i^f the root of f-item $T(i)$, and by r_i^g the leaf (non-empty) child of bridge item $T(i)$. (Also present in this figure are a number of "g-items", which are not part of the f-100 tree but are attached to it in a manner to be described.)

Figure 3. An f-100-tree

There are two reasons for letting the bridge items of an f-i-tree have the empty string as one of their children. First, it makes them de facto f-items with only one child, a subtle point in our proof of security that is pointed out in remark 1. Second, it makes them distinguishable from items with two children, a simple point used, for instance, in Lemma 2.

8. DESCRIPTION OF OUR SIGNATURE SCHEME

8.1 Message Spaces

The security properties of the new signatures scheme hold for any nonempty message space $M \subset \{0,1\}^+$.

8.2 How to Generate Keys

We assume the existence of a claw-free permutation pair generator G and, without loss of generality, that the bound B on the number of signatures that can be produced is a power of 2: $B = 2^b$.

The key-generation algorithm K runs as follows on inputs 1^k and 2^b:

(1) K runs G twice on input 1^k to secretly and randomly select two quintuples

$$(d_f, f_0, f_0^{-1}, f_1, f_1^{-1}), \text{ and } (d_g, g_0, g_0^{-1}, g_1, g_1^{-1}) \in [G(1^k)].$$

(2) K then randomly selects r_ϵ^f in $D_f = [d_f()]$.

(3) K outputs the public key $PK = (f, r_\epsilon^f, g, 2^b)$ where f is the claw-free pair (d_f, f_0, f_1) and g is the claw-free pair (d_g, g_0, g_1).

(4) K outputs the secret key $SK = (f^{-1}, g^{-1})$.

The PK and SK so produced are said to be *(matching) keys of size k.*

8.3 What Is a Signature

A *signature* of a message m with respect to a public key $(f, r_\epsilon^f, g, 2^b)$ consists of:

(1) An f-chain of length $b+1$ starting at a string r_ϵ^f and ending at r^g, and

(2) A g-item with r^g as its root and m as its only child.

8.4 How To Sign?

In the remainder of this section we shall presuppose that user A's public key is $PK = (f, r_\epsilon^f, g, 2^b)$ where $f = (d_f, f_0, f_1)$ and $g = (d_g, g_0, g_1)$. User A's secret key is $SK = (f^{-1}, g^{-1})$. We denote by D_f the domain $[d_f()]$, and denote by D_g the domain $[d_g()]$ similarly.

Conceptually, user A creates an f-1^b-tree T, which has 2^b leaves. The root of T will be r_ϵ^f. The other internal nodes of T are randomly selected elements of D_f. The leaves of T are randomly selected elements of D_g.

To sign m_i, the $i-th$ message in the chronological order, user A computes a g-item G_i whose root $r_i^g \in D_g$ is the ith leaf of T, and whose only child is the message m_i. He then outputs, as the signature of m_i, G_i and the f-chain in T starting at root r_ϵ^f and ending at leaf r_i^g.

In practice, it will be undesirable for user A to precompute and store all of T. He will instead "grow" T as needed and try to optimize his use of storage and time. This is taken into account by our signing procedure. In what follows, we describe a variation of our signing method that requires the signer to remember just his secret key and his most recently produced signature, in order to produce his next signature. The reader may find it helpful to refer to Figure 3 while reading this description.

The Signing Procedure (also called $\mathcal{SP}$):

We presume that the procedure is initialized with the values of the public key PK and the corresponding secret key SK in its local private storage, that has already signed messages $m_0, m_1, \ldots, m_{i-1}$ and kept track of the number of previous messages signed (i.e. the variable $i = i_0 \ldots i_{b-1}$ which is a b-long bit string, which may contain leading 0's), and the most recent signature produced.

To compute a signature for message m_i, the i-th message, user A performs the following steps.

(1) *(Output f-chain.)*

(1.1) *(Output f-items in common with previous signature.)* If $i = 0^b$ this substep is skipped, and control passes to step (1.2). Otherwise, for each string j which is a common prefix of i and $i-1$, he outputs the f-item $(t_j^f, r_j^f; r_{j0}^f, r_{j1}^f)$ which was part of the signature for message m_{i-1}, in order of increasing length of j.

(1.2) *(Output new f-items in f-tree.)* For each string j (if any) which is a proper prefix of i, but not a prefix of $i-1$, user A creates and outputs an f-item $T(j)$, in order of increasing length of j. The f-item $T(j) = (t_j^f, r_j^f; r_{j0}^f, r_{j1}^f)$ is created as follows: If $j = \epsilon$ its root r_j^f is the r_ϵ^f from the public key; otherwise it is the k-th child of the most recently output f-item, where k is the last bit of the string j. The children r_{j0}^f and r_{j1}^f of the f-item with root r_j^f are chosen at random from D_f. The tag $t_j^f = f^{-1}_{\langle r_{j0}^f, r_{j1}^f \rangle}(r_j^f)$ is computed using f_0^{-1} and f_1^{-1} from the secret key. Note that the last item output (by either step (1.1) or (1.2)) has r_i^f as one of its children.

(1.3) *(Output bridge f-item.)* User A next outputs a single f-item with root r_i^f and whose children are ϵ and r_i^g, a randomly chosen element from D_g. The tag t_i^f for this item is again computed using the secret trap-door information for inverting f_0 and f_1.

(2) *(Output g-item.)* Finally, user A outputs the g-item $G_i = (t_i^g, r_i^g; m_j)$. The tag t_i^g for this item is computed using the g^{-1} from the secret key.

The items output by the above procedure constitute a signature for m_i. Notice that there are many possible signatures (among which A chooses one at random) for each occurrence of each message, but only one signature is actually output.

The reader may verify that the above procedure for producing a signature will have a total running time which is bounded by a polynomial in k and b.

Notice that if A has signed i messages, the function T mapping each string $j \in DFS(i)$ to f-item $T(j)$ is an f-i-tree as defined in section 7.

8.5 How to Verify a Signature

Given A's public key $(f, r_\epsilon^f, g, 2^b)$, anyone can easily verify that the first $b+1$ elements in the signature of m_i are f-items forming an f-chain starting at r_ϵ^f and ending at r_i^g, and that the g-item in the signature has r_i^g as its root and m_i as its only child. If these checks are all satisfied, the given sequence of items is accepted as an authentic signature by A of the message m_i.

It is easy to confirm that these operations take time proportional to b times some polynomial in k, the size of the public key.

8.6 Efficiency of the Proposed Signature Scheme

Assume that if $f = (d_f, f_0, f_1)$ is a claw-free pair of size k, then an element of D_f is specified by a k-bit string. Then the time to compute a signature for a message m of length l is is $O(bk)$ f-inversions (i.e. inversions of f_0 or f_1) and $O(l)$ g-inversions.

Another relevant measure of efficiency is “amortized” time. That is, the time used for producing all possible 2^b signatures divided by 2^b. In our scheme, the amortized “f-inversion” cost is $O(k)$. The amortized “g-inversion” cost is $O(l)$ if the average length of a message is l.

The length of the signature for m is $O(bk+l)$, where l is the length of m, as m is included in m's signature as the child of the g-item. Clearly, if m is known to the signature recepient, the g-item need not include m: it suffices to give its root and its tag. This way the length of the signature can be only $O(bk)$ long, which is independent of the length of m and possibly much shorter.

The memory required by the signing algorithm is $O(bk)$ since it consists of storing (the f-items in) the most recently produced signature.

9. PROOF OF SECURITY

Let us start by establishing a convenient terminology.

Definition: We call *signature corpus* the first i (for some $i > 0$) signatures output by our signing procedure $\mathcal{SP}$. We shall generally use the symbol $\mathcal{S}$ to denote a signature corpus.

We define the following quantities relative to a signature corpus $\mathcal{S}$, consisting of i signatures relative to a public key $PK = (f, r_\epsilon^f, g, 2^b)$.

(1) The set of *items* of $\mathcal{S}$, denoted by $\mathcal{I}(\mathcal{S})$, is the set of the items in the signatures of $\mathcal{S}$.
(2) The set of *f-items* of $\mathcal{S}$, denoted by $f(\mathcal{S})$, is the set of f-items in $\mathcal{I}(\mathcal{S})$.
(3) The set of *g-items* of $\mathcal{S}$, denoted by $g(\mathcal{S})$, is the set of g-items in $\mathcal{I}(\mathcal{S})$.
(4) The set of *messages* of $\mathcal{S}$, denoted $M(\mathcal{S})$, is the set of messages signed by $\mathcal{S}$, i.e. the set of children of the g-items of $\mathcal{S}$.
(5) The *f-tree* of $\mathcal{S}$, denoted by $\mathcal{T}^f(\mathcal{S})$, is the f-i-tree having root r^f_ϵ and, as path to leaf j ($j = 0, ..., i$), the f-chain of the j-th signature of $\mathcal{S}$.
(6) The set of *internal nodes* of $\mathcal{S}$, denoted by $\mathcal{IN}(\mathcal{S})$, is the set of the internal nodes of $\mathcal{T}^f(S)$.
(7) The set of *non-roots* of $\mathcal{S}$, denoted by $\mathcal{NR}(\mathcal{S})$, is the set of those internal nodes of $\mathcal{T}^f(\mathcal{S})$ that are not the root of any f-item of $\mathcal{S}$. We may think of these nodes as "hooks" from which additional f-items will be grown as new signatures are created.
(8) The set of *leaves* of $\mathcal{S}$, denoted $\mathcal{L}(\mathcal{S})$, is the set of leaves of $\mathcal{T}^f(\mathcal{S})$.

Notice that all the above sets are unambiguously defined. For instance, an item in $f(\mathcal{S})$ has exactly two children while an item in $g(\mathcal{S})$ only one, the bridge elements of $\mathcal{I}(\mathcal{S})$ have exactly one empty child and thus are distinguishable from other items in $f(\mathcal{S})$, and so on.

Some of these definitions can be observed in figure 3. For example, the leaves of the f-101-tree in figure 3 are $r^f_{000}, r^f_{001}, r^f_{010}, r^f_{011}, r^f_{100}$ and its non-roots are r^f_{101}, and r^f_{11}.

Let us now see how the signature of a message never signed before relates to a given signature corpus.

Lemma 2: Let $\mathcal{S}$ be a signature corpus relative to a public key $PK = (f, r^f_\epsilon, g, 2^b)$ and let σ be a signature (relative to the same public key) of a message m not belonging to $M(\mathcal{S})$. Denote by $\mathcal{I}(\sigma)$ the set of items in σ. Then $\mathcal{I}(\sigma) - \mathcal{I}(\mathcal{S})$ (the set of new items) contains either
(1) a g-item with root $r \in \mathcal{L}(\mathcal{S})$ or
(2) an f-item with root $r \in \mathcal{IN}(\mathcal{S})$.

Proof: First notice that $\mathcal{I}(\sigma) - \mathcal{I}(\mathcal{S})$ is not empty as it contains G, the g-item of σ. In fact, G cannot belong to $f(\mathcal{S})$, as it is a g-item, and cannot belong to $g(\mathcal{S})$, as m is its only child and all items in $g(\mathcal{S})$ have elements of $M(\mathcal{S})$ as their children. Assume $\mathcal{I}(\sigma) - \mathcal{I}(\mathcal{S})$ also contains an f-item. Then this f-item belongs to F, the f-chain of σ whose first item has r^f_ϵ as root, one of the internal nodes of $\mathcal{S}$. Thus, for some item in F, (2) holds. Assume now that $\mathcal{I}(\sigma) - \mathcal{I}(\mathcal{S}) = G$. Then the root of G is the non-empty child of B, the bridge f-item of σ. By hypothesis B is in $\mathcal{I}(\mathcal{S})$, thus the root of G belongs to $\mathcal{L}(\mathcal{S})$ and (1) holds.

∎

Recall lemma 1 from section 6.4.

Lemma 1: Let $f = (d, f_0, f_1)$ be a claw-free pair, x and y be elements of d and i, j two different tuples of binary strings such that there exists a string z such that $z = f_{\langle i \rangle}(x) = f_{\langle j \rangle}(y)$. Then there exists an f-claw (x_1, x_2, x_3) where $x_3 = f_c^{-1}(z)$ for some prefix c of $\langle i \rangle$.

We can now prove Lemma 3.

Lemma 3: There exists a polynomial-time algorithm A that, on input a corpus $\mathcal{S}$ relative to a public key $PK = (f, r^f_\epsilon, g, 2^b)$ and the signature σ of a message not belonging to $M(\mathcal{S})$, finds either
(1) a g-claw or
(2a) an f-claw or
(2b) an f-item whose root belongs to $\mathcal{NR}(\mathcal{S})$.

Proof: (the cases are numbered according to the corresponding cases in Lemma 2)

If case (1) of Lemma 2 holds for $\mathcal{S}$ and σ, then we have two g-items with the same root r in $\mathcal{L}(\mathcal{S})$. Namely, an i, j, x and y such that $g_{\langle i \rangle}(x) = g_{\langle j \rangle}(y) = r$ and we get a g-claw by Lemma 1. Otherwise, if case (2) of lemma 2 holds, let F be the f-item that satisfies condition (2) of Lemma 2. If F has the same root as some $F' \in f(\mathcal{S})$, then again by Lemma 1, we get an f-claw, otherwise we get an f-item whose root belongs to $\mathcal{NR}(\mathcal{S})$.

∎

Remark 1: Notice that if σ is generated by the legal signer (i.e. the $\mathcal{SP}$ procedure) then, with very high probability, case (2b) will hold in lemma 3.

In the proof of the main theorem we will assume that there exists a successful adaptive chosen-message attack, and derive a contradiction by showing that this attack would enable an enemy to easily create either an f-claw or a g-claw with sufficiently high probability. Recall that in an adaptive chosen-message attack the enemy can repeatedly use the real signer as an "oracle" before attempting to forge a new signature. The next lemma, (lemma 4), essentially states that the signing process can be simulated perfectly by an efficient algorithm that knows the public key and only half of the secret key: the inverses of the first claw-free pair. (i.e. in some sense, this algorithm is a forger.)

To state lemma 4, additional notation regarding "interactive" probabilistic algorithms, needs to be introduced. The notion of an adaptive, chosen-message attack involves the interaction of two algorithms: $\mathcal{SS}$ (the signer) and $\mathcal{SR}$ (the signature requestor). These algorithms "take turns": $\mathcal{SR}$ requests a signature of a given message, $\mathcal{SS}$ signs it, $\mathcal{SR}$ requests a second signature, $\mathcal{SS}$ computes it, and so on. We might view the two routines as "co-routines" that pass control back and forth while preserving their own state. We formalize this interaction by means of the *combining algorithm* $\mathcal{C}$ that defines a composite algorithm from two auxiliary ones. The combining algorithm $\mathcal{C}$ will invoke repeatedly $\mathcal{SS}$ and $\mathcal{SR}$ in alternation, corresponding to their taking turns. The algorithms $\mathcal{SS}$ and $\mathcal{SR}$ have private state variables (denoted $V_{\mathcal{SS}}$ and $V_{\mathcal{SR}}$) that are preserved from invocation to invocation. Algorithm $\mathcal{SS}$ (which produces signatures) takes as input a public key PK, an auxilary input X (which for the moment is unsepecified but will later denote either the corresponding secret key SK or part of it), a new message to sign, and its private state variable. It produces as output a signature for the new message and an updated version of its private state variable. Similarly, $\mathcal{SR}$ is a probabilistic algorithm which takes as input a public key, a sequence of previous signatures relative to that public key, and its private state variable, and produces as output a message to be signed and an updated version of its private state variable.

The following algorithm makes specific the process of combining $\mathcal{SS}$ and $\mathcal{SR}$:

Algorithm $\mathcal{C}(\mathcal{SS}, \mathcal{SR}; PK, X, i)$

Set $\mathcal{S}_0 \leftarrow \phi$.
Set $V_{\mathcal{SR}}$ and $V_{\mathcal{SS}}$ to ϕ.
for $j = 0$ to i do:
$(m_j, V_{\mathcal{SR}}) \leftarrow \mathcal{SR}(PK, \{\mathcal{S}_1, \ldots, \mathcal{S}_{j-1}\}, V_{\mathcal{SR}})$ *(Request signature for message m_j.)*
$(\mathcal{S}_j, V_{\mathcal{SS}}) \leftarrow \mathcal{SS}(PK, m_j, V_{\mathcal{SS}}, X)$. *(Produce signature for message m_j.)*
Output $\mathcal{S}_j$.

Here $\mathcal{S}_j$ denotes the signature of the j-th message.

We extend our notation of probabilistic algorithm in a natural way by letting $\mathcal{C}(\mathcal{SS}, \mathcal{SR}; PK, X, i)$ represent the probability space that assigns the sequence σ the probability that $\mathcal{C}$ outputs σ after invoking alternatively (for i times) $\mathcal{SS}$ (with initial input PK and X) and $\mathcal{SR}$ (with initial inputs PK).

We can now state lemma 4, stating that the signing process can be simulated effectively if the f_i's inverses are known but the g_i inverses are not.

Lemma 4: There exists an algorithm $\mathcal{A}$ in $\mathcal{RA}$ such that for all requestors $\mathcal{SR} \in \mathcal{RA}$, for all public keys $PK = (f, r^f_\epsilon, g, 2^b)$ and for all non-negative integers $i < 2^b$,

$$\mathcal{C}(\mathcal{A}, \mathcal{SR}; PK, \{f^{-1}\}, i) = \mathcal{C}(\mathcal{SP}, \mathcal{SR}; PK, SK, i)$$

(Where $\mathcal{SP}$ is the legal signing process of section 8, and SK is the corresponding secret key to PK).
Proof. Consider the following algorithm $\mathcal{A}$. We inductively assume that

$$\mathcal{C}(\mathcal{A}, \mathcal{SR}; PK, \{f^{-1}\}, i-1) = \mathcal{C}(\mathcal{SP}, \mathcal{SR}; PK, SK, i-1)$$

Thus the f-chains in the first $i-1$ signatures output by $\mathcal{C}$ uniquely define an f-$(i-1)$-tree T. Algorithm $\mathcal{A}$ stores $i-1$ and the f-chain of the last produced signature and executes the following instructions to sign m_i, the i-th message, where $i{=}i_0 \cdots i_b$.

(1) *(Authenticate m_j with a g-item.)* Pick an element t^g_j at random in D_g and compute $r^g_j = g_{\langle m_j \rangle}(t^g_j)$ so to generate the g-item $(t^g_j, r^g_j; m_j)$

(2) *(Build the f-chain from r^f_ϵ to r^f_j to the extent that it is not already done.)* Compute $i_0 i_1 \cdots i_j$, the longest proper prefix of i that is also a prefix of $i-1$. For $x = 1$ to $b-j$, generate $T(i_0 \cdots i_{j+x})$, an

f-item whose root is the i_{j+x}-th child of $T(i_o \cdots i_{j+x-1})$, and whose two children are independently and randomly selected elements of D_f. (Algorithm $\mathcal{A}$ easily computes the tag of this new f-item by using f^{-1}.)

(3) *(Create the bridge item authenticating r_j^g.)* Using f_0^{-1} and f_1^{-1}, create a f-item with children ϵ and r_i^g and having as root the i_b-th child of $T(i_0 \cdots i_{b-1})$.

(4) *(Output signature of m_i)* Output $T(\epsilon), T(i_0), \ldots T(i_0 \cdots i_{b-1})$, the new bridge item $T(i_0, ..., i_b)$ and the new g-item. ∎

In lemma 6 we show a similar result: the signing process can be simulated if g^{-1} is known, but f^{-1} is not. The proof of lemma 6 makes essential use of the fact that there is a known upper bound on the number of signatures to be produced. (The bound provides a limit on the amount of a preprocessing step that is the subject of lemma 5.)

There is, however, a very important difference between the signing simulation procedure described in lemma 4 (which uses f^{-1} but not g^{-1}) and that of lemma 6 (which uses g^{-1} but not f^{-1}). The proof of lemma 4 works with any fixed root r_ϵ^f, which can be fixed arbitrarily before the simulation procedure is invoked.

By contrast, the signing simulation procedure of lemmas 5 and 6 actually *produces* the necessary root r_ϵ^f to be part of the public key in its preprocessing step. The root produced is uniformily distributed over D_f. Thus, from the point of view of an observer that monitors the behaviour of the signer when he publishes his public key, the preprocessing step is undistinguishable from a genuine key generation step. Moreover, by monitoring the signing process, the observer can not tell whether the signer really knows f^{-1} or he has first applied the preprocessing procedure of lemma 5 to produce his public file and only then applied the simulation procedure of lemma 6.

Definition: For all strings $m_1, ..., m_i$, let *sequence*$(m_1, ..., m_i)$ denote the trivial interactive algorithm that, no matter what inputs it gets, when invoked for the j-th time ($j = 1, ..., i$) outputs the string m_j.

Let us define two probability spaces over the f-i-trees which are crucial to our analysis.

Definition: Let $PK = (f, r_\epsilon^f, g, 2^b)$ and $SK = (f^{-1}, g^{-1})$ be a pair of matching public and secret key, where $f = (d_f, f_0, f_1)$. Recall that $\mathcal{C}$ is the combining algorithm. Define two probability spaces, $\mathcal{T}_{i,PK}$ and $\mathcal{T}_{i,f,g,2^b}$, as follows:

$\mathcal{T}_{i,PK}$ is generated by randomly selecting $\mathcal{S}$ in $\mathcal{C}(\mathcal{SP}, sequence(m_1, ..., m_i); PK, SK, i)$ and then computing $\mathcal{T}^f(\mathcal{S})$. (Note that $\mathcal{T}_{i,PK}$ does not depend on the values of the messages $m_1, \ldots, m_i$ but it does depend on i, the number of messages.)

$\mathcal{T}_{i,f,g,2^b}$ is generated by randomly selecting $\mathcal{S}$ in $\mathcal{C}(\mathcal{SP}, sequence(m_1, ..., m_i); (f, d_f(), g, 2^b), SK, i)$ and then computing $\mathcal{T}^f(\mathcal{S})$.

Informally, $\mathcal{T}_{i,PK}$ is the probability space obtained from $\mathcal{T}_{i,f,g,2^b}$ by randomly picking $r_\epsilon^f \in D_f$ and fixing it in PK.

Notice that both probability spaces are easily generated if the secret key $SK = (f^{-1}, g^{-1})$ is among the available inputs. However, both probability spaces remain easy to generate on a more restricted set of inputs. It has been implicitly proved in lemma 2 that $\mathcal{T}_{i,PK}$ can be generated in probabilistic polynomial-time on inputs i, PK and f^{-1} alone. The following lemma shows that $\mathcal{T}_{i,f,g,2^b}$ is easily generated on inputs $i, f, g, 2^b$ alone.

Lemma 5: There exists $\mathcal{T} \in \mathcal{RA}$ such that for all claw-free pairs $f = (d_f, f_0, f_1)$ and $g = (d_g, g_0, g_1)$ and for all integers $i < 2^b$,

$$\mathcal{T}(i, f, g, 2^b) = \mathcal{T}_{i,f,g,2^b}.$$

Proof: Consider the following algorithm $\mathcal{T}$ that constructs an f-i-tree T in "reverse order"; that is, it constructs f-item $T(x)$ before f-item $T(y)$ if $y < x$. (This is necessary since $\mathcal{T}$ does not have access to f^{-1}.) The construction goes as follows.

If string $j \in DFS(i)$ has length b, $\mathcal{T}$ selects the non-empty child of $T(j)$ at random in D_g. Otherwise (if j has length shorter than b), $\mathcal{T}$ selects, as 0-th child of $T(j)$, the root of $T(j0)$ and, as 1st child, the root of $T(j1)$. In case $j1$ does not belong to $DFS(i)$, $\mathcal{T}$ selects the second child of $T(j)$ at random in D_f.

Having selected the two children c_o and c_1 of $T(j)$, $\mathcal{T}$ selects its tag t at random in D_f. Then it computes the prefix-free encoding $\langle(c_0, c_1)\rangle$ and selects as the root of $T(j)$ the element $f_{\langle c\rangle}(t)$, which $\mathcal{T}$ easily computes using f_0 and f_1.

Notice that each $T(j)$ so computed is a proper f-item and that the resulting T is a proper f-i-tree belonging to $[\mathcal{T}_{i,f,g,2^b}]$. Let's now analyse the probability distribution according to which T has been selected.

First notice that the leaves of T (that is the non-empty children of the items of depth b) have the same distribution of the leaves of a f-i-tree randomly selected in $\mathcal{T}_{i,f,g,2^b}$. In fact, in both cases, all leaves are uniformily and independently selected elements of D_g. Then notice that the roots of the items of T of depth k (that is the children of the items of T of depth $k-1$) are selected uniformly and independently in D_f. In fact, the root of each item is obtained by applying $f_{\langle x\rangle}$, a permutation of D_f randomly selected from some probability space, to an element t (the tag) *independently* and uniformily selected in D_f. From this it easily follows that $\mathcal{T}$ selects T at random in $\mathcal{T}_{i,f,g,2^b}$. It is easily seen that $\mathcal{T} \in \mathcal{RA}$ and thus satisfies all the required properties of our lemma. ∎

Lemma 6: There exists an algorithm $\mathcal{A} \in \mathcal{RA}$ such that for all signature requestors $\mathcal{SR} \in \mathcal{RA}$, for all claw-free pairs $f = (d_f, f_0, f_1)$ and $g = (d_g, g_0, g_1)$, and for all non-negative integers $i < 2^b$,

$$\mathcal{C}(\mathcal{A}, \mathcal{SR}; (f, d_f(), g, 2^b), \{g^{-1}\}, i) \;=\; \mathcal{C}(\mathcal{SP}, \mathcal{SR}; (f, d_f(), g, 2^b), \{f^{-1}, g^{-1}\}, i).$$

Proof: Consider the following algorithm $\mathcal{A}$. In a preprocessing step, $\mathcal{A}$ runs algorithm $\mathcal{T}$ of Lemma 5 to randomly select an f-i-tree T from $\mathcal{T}_{i,f,g,2^b}$. Let r^f_ϵ be the root of T. This root is used to construct the public file $PK = (f, r^f_\epsilon, g, 2^b)$, with respect to which all subsequent signatures will be produced as follows. $\mathcal{A}$ starts the signature requestor $\mathcal{SR}$ on input PK. Then it simulates the signing procedure with initial inputs PK and the corresponding secret key $SK = (f^{-1}, g^{-1})$ without using f^{-1} in the following way. When $\mathcal{SR}$ outputs m_j, the j-th message to be signed, $\mathcal{A}$ retrieves the f-chain T_j, the path from the root of T to leaf j. Then $\mathcal{A}$ computes the necessary g-item by using g^{-1}. ∎

Before stating and proving our main theorem, let us single out a simple lemma stating that one cannot invert a claw-free pair on a randomly selected input of its domain.

Lemma 7: Let G be a claw-free permutation pair generator. Then, for any inverting algorithm $\mathcal{I} \in \mathcal{RA}$, any $c > 0$ and sufficiently large k,

$$\mathbf{P}(h_0(z) = x \textit{ or } h_1(z) = x | (d, h_0, h_0^{-1}, h_1, h_1^{-1}) \leftarrow G(1^k); x \leftarrow d_h(); z \leftarrow I(1^k, d, h_0, h_1)) < k^{-c}.$$

Proof: Otherwise the following algorithm would find a claw with too high a probability: randomly select y in d_h, randomly select i between 1 and 2, compute $x = h_i(y)$ and run I to get z such that $h_j(z) = x$ for $j \neq i$. ∎

We are now ready to formally state and prove our main theorem. We start by strengthening the definition of existentially forgeable to include probabilistic success on the part of the forger.

Definition: We say that a signature scheme is *ϵ-existentially forgeable* if it is existentially foregeable with probability ϵ where the probability space includes the random choices of the adaptive chosen-message attack, the random choices made by the legal signer in the creation of the public key, and the random choice made by the legal signer in producing signatures.

It is very important to note that the random choices made in creating the public key are included in the probability space; our proof depends critically on this definition. The main theorem of this paper is the following.

Main Theorem. Assuming that claw-free permutation pair generators exist, the signature scheme described in section 8 is not even $\frac{1}{Q(k)}$-existentially forgeable under an adaptive chosen-message attack, for all polynomials Q and for all sufficiently large k.

Proof of the Main Theorem. The proof proceeds by contradiction. We assume, for contradiction sake, that for some polynomial Q and for infinitely many k our signature scheme is $\frac{1}{Q(k)}$-existentially forgeable under an adaptive chosen message attack by an algorithm $\mathcal{F}$ in $\mathcal{RA}$.

By definition, the forging algorithm $\mathcal{F}$ consists of two algorithms in $\mathcal{RA}$: a signature requestor $\mathcal{FR}$, which is active in a first phase when it adaptively asks and receives signatures of messages of its choice, and a signature finder $\mathcal{FF}$, which is active in a second phase when it attempts to forge a signature of a message not asked about by $\mathcal{FR}$.

Let $PK = (f, r_\epsilon^f, g, 2^b)$ and SK be a public/secret-key pair of size k, randomly selected by our key generator using a claw-free permutation pair generator G. In the first phase a signature corpus $\mathcal{S} \leftarrow \mathcal{C}(\mathcal{SP}, \mathcal{FR}; PK, SK, i)$ is generated, where $i < 2^b$. Then $\mathcal{FF}$ is run on input $\mathcal{S}$ and PK. Let ϵ_k denote the probability that $\mathcal{FF}$ outputs σ, a legal signature, with respect to PK, for a message $m \notin M(\mathcal{S})$. (This probability is taken over all the coin tosses of G, $\mathcal{FR}$, $\mathcal{FF}$ and $\mathcal{SP}$).

What we have assumed is that, for infinitely many k,

$$\epsilon_k \geq \frac{1}{Q(k)}.$$

By Lemma 3, given $\mathcal{S}$ and σ, it is now easy to compute either
(1) a g-claw (i.e. a claw for the second claw-free pair in PK) or
(2) an f-claw (i.e. a claw for the first claw-free pair in PK) or
(3) an f-item whose root belongs to $\mathcal{NR}(\mathcal{S})$.

Denote the probability that case (1), (2) or (3) hold respectivey by δ_1, δ_2 and δ_3. Then, for infinitely many k, we have

$$\delta_1(k) + \delta_2(k) + \delta_3(k) \geq \epsilon_k > \frac{1}{Q(k)}.$$

Thus either
(1') there is an infinite set K_1 so that for $k \in K_1$ $\delta_1(k) > \frac{1}{3Q(k)}$, or
(2') there is an infinite set K_2 so that for $k \in K_2$ $\delta_2(k) > \frac{1}{3Q(k)}$, or
(3') there is an infinite set K_3 so that for $k \in K_3$ $k \in K_3$ $\delta_3(k) > \frac{1}{3Q(k)}$.

We will show that either case leads to contradiction.

Assume case (1') holds. Then consider the following algorithm in $\mathcal{RA}$ that, on input 1^k and a claw-free pair $h = (d_h, h_0, h_1)$ of size k randomly selected by G, finds an h-claw with sufficiently high probability.

Algorithm 1: Run G on input 1^k to randomly select a quintuple $(d_f, f_0, f_0^{-1}, f_1, f_1^{-1})$. Select $r_\epsilon^f \in D_f$ at random and construct the public key $PK = (d_f, f_0, f_1, r_\epsilon^f, d_h, h_0, h_1, 2^b)$. (Notice that PK is a random public key of size k of our signature scheme.) Randomly select the signature corpus $\mathcal{S} \leftarrow \mathcal{C}(\mathcal{SP}, \mathcal{FR}; PK, SK, i)$. Though PK's matching secret key SK is not totally known, this random selection can be efficiently done as, by Lemma 4, there exists an $\mathcal{A} \in \mathcal{RA}$ such that $\mathcal{C}(\mathcal{SP}, \mathcal{FR}; PK, SK, i) = \mathcal{C}(\mathcal{A}, \mathcal{FR}; PK, f^{-1}, i)$. Now run $\mathcal{FF}$ on input $\mathcal{S}$ and PK to sign a new message. From this last signature and $\mathcal{S}$, try to compute an h-claw.

Notice that, for $k \in K_1$, Algorithm 1 will successfully compute an h-claw with probability $\delta_1(k) > \frac{1}{3Q(k)}$. This contradicts the claw-freeness of G.

Assume now that either (2') or (3') hold. Consider the following algorithm in $\mathcal{RA}$, whose input is 1^k and a claw-free pair $h = (d_h, h_0, h_1)$ of size k randomly selected by G.

Algorithm 2: Run G on input 1^k to randomly select a quintuple $(d_g, g_0, g_0^{-1}, g_1, g_1^{-1})$. Randomly select the signature corpus

$$\mathcal{S} \leftarrow \mathcal{C}(\mathcal{SP}, \mathcal{SR}; (h, d_h(), g, 2^b), \{h^{-1}, g^{-1}\}, i)$$

which can be done as by lemma 6 there exists an algorithm $\mathcal{A} \in \mathcal{RA}$ such that

$$\mathcal{C}(\mathcal{SP}, \mathcal{SR}; (h, d_h(), g, 2^b), \{h^{-1}, g^{-1}\}, i) = \mathcal{C}(\mathcal{A}, \mathcal{SR}; (h, d_h(), g, 2^b), g^{-1}, i)$$

. Then run $\mathcal{FF}$ on input $\mathcal{S}$ and PK.

Assume that case (2') holds. Then, for $k \in K_2$, from the output of Algorithm 2 an h-claw can be computed with sufficiently high probability to violate the claw-freeness of G.

Finally, assume that case (3') holds and $k \in K_3$. Then, given a random $x \leftarrow d_h()$, the following algorithm $\mathcal{I}$ will invert h on x with non-negligible probability (contradicting Lemma 7). $\mathcal{I}$ runs Algorithm 2 except that, when constructing $\mathcal{T}^h(\mathcal{S})$ as in Lemma 5, makes x the value of a randomly selected non-root of $\mathcal{S}$. Notice that this operation does not change the probability distribution of $\mathcal{S}$. (Recall that the pre-processing procedure of Lemma 5 just picks at random all the internal nodes of $\mathcal{S}$.) Thus $\mathcal{S}$ is a random signature corpus with respect to a randomly selected public key of size k. Thus, from the output of Algorithm 2, $\mathcal{I}$ computes an h-item with root $r \in \mathcal{NR}(\mathcal{S})$ with probability $\delta_3(k) > \frac{1}{3Q(k)}$. When this happens, with probability $\frac{1}{|\mathcal{NR}(\mathcal{S})|}$ we have $r = x$. Now, given the h-item computed, $\mathcal{I}$ can easily compute either $h_0^{-1}(x)$ or $h_1^{-1}(x)$, and lemma 7 is contradicted. This completes the proof of the main theorem. ∎

10. VARIATIONS AND IMPROVEMENTS

In this section we describe ways to improve the efficiency of the proposed signature scheme without affecting its security.

10.1 Using g_i's to sign rather than g_i^{-1}'s.

This variation is of interest if it is substantially easier to compute g_0 or g_1 than to compute their inverses. In this case steps (3) and (4) in the signing procedure can be replaced by:

(3) *(Output g-item.)* User A selects a random $t_i^g \in D_g$, and (using g_0 and g_1) computes the root r_i^g of the g-item $(t_i^g, r_i^g; m_i)$, and outputs this item.

(4) *(Output bridge f-item.)* Using his knowledge of f_0^{-1} and f_1^{-1}, user A outputs an f-item with root r_i^f and an only child r_i^g.

Now each usage of g_0^{-1} or g_1^{-1} has been replaced by a usage of g_0 or g_1.

Although one might be tempted to use this variation using one-way permutations instead of trap-door permutations for the g_i's, this temptation should be resisted, since our proof of security does not hold if this change is made.

10.2 Fast iterated square roots

As we saw in section 6.3, if factoring is computationally hard, a particular family of trap-door permutations is claw-free. By using these permutations in a straightforward manner, one obtains a particular instance of our signature scheme. Let us discuss its efficiency of this instance. The computation of $f_0^{-1}(x)$ consists of computing the square-root which has Jacobi symbol 1 and is less than $n/2$, modulo a Blum-integer n. We can compute $f_1^{-1}(x)$ as $f_0^{-1}(x/4)$. Computing $g_0^{-1}(x)$ and $g_1^{-1}(x)$ is the same, except for using the appropriate n. If n is k-bits long, this can be done in $O(k^3)$ steps. Thus the signature of a k-bit message can be computed in time $O(b \cdot k^4)$, or in $O(k^4)$ amortized time.

This particular instance of our scheme can be improved in a manner suggested in discussions with Oded Goldreich (see [Go86] – we appreciate his permission to quote these results here). The improvement relates to the computation of $f_{\langle y \rangle}^{-1}(x)$ (or $g_{\langle y \rangle}^{-1}(x)$).

We note first of all that taking square roots modulo n is equivalent to taking u-th powers modulo n, where $u \cdot 2 \equiv 1 \pmod{\phi(n)}$, and where $\phi(n)$ is Euler's phi function. More generally, to find a 2^m-th root w of x modulo n one can raise x to the v-th power modulo n, where $v \equiv u^m \pmod{\phi(n)}$. Computing w by first computing v and then raising x to the v-th power is substantially faster than repeatedly taking square roots.

To apply this observation, we note that the functions f defined in section 6.3. satisfy

$$f_{\langle y \rangle}^{-1}(x) = \pm\left(\frac{x}{4^{rev(\langle y \rangle)}}\right)^{2^{-m}},$$

where "rev" is the operation which reverses strings and interprets the result as an integer, where m is the length of $\langle y \rangle$, where all operations are performed modulo n, and where the final sign is chosen to make the result less than $n/2$. The only computationally difficult portion here is computing a 2^m-th root. Using the observation of the previous paragraph, the computation of such an f-inverse can be performed in time proportional to the cube of the length of n, in the case that messages have the same length k as n. Using these ideas, the signature of a k-bit message can be computed in time $O(b \cdot k^3)$, or in $O(k^3)$ amortized time.

10.3 "Memoryless" Version of the Proposed Signature Scheme

The concept of a *random function* was introduced by Goldreich, Goldwasser and Micali in [GGM84].

Let I_k denote the set of k-bit integers. Let W_k denote the set of all functions from I_k to I_k, and let $F_k \subseteq W_k$ be a set of functions from I_k to I_k. We say that $F = \bigcup_k F_k$ is a *poly-random* collection if:

(1) Each function in F_k has a unique k bit index associated with it. Furthermore, picking such an index at random (thereby picking an $f \in F_k$ at random) is easy.

(2) There exists a deterministic polynomial time algorithm that given as input an index of a function $f \in F_k$ and an argument x, computes $f(x)$.

(3) No probabilistic polynomial in k time algorithm can "distinguish" between W_k and F_k. Formally, let T be a probabilistic polynomial time algorithm, that on input k and access to an oracle O_f for a function $f : I_k \rightarrow I_k$ outputs 0 or 1. Then, for all T, for all polynomials Q, for all sufficiently large k, the difference between the probability that T outputs 1 on access to an oracle O_f when f was randomly picked in F_k and the probability that T outputs 1 on access to an oracle O_f when f was randomly picked in W_k is less than $1/Q(k)$.

In [GGM84] it was shown how to contruct a poly-random collection assuming the existence of one-way functions. The existence of claw-free permutation pairs is a stronger assumption, and thus implies the existence of a poly-random collection. See section 5.4 for an implementation of a claw-free family of functions based on factoring and [GGM84] for details on how to contruct a poly-random collection.

Leonid Levin suggested the following use of a poly-random collection in order to reduce the amount of storage that a signer must keep from $O(bk)$ to $O(b)$ bits. His suggestion also eliminates the need to generate new random numbers (e.g. r_i^g) during the signing process.

Let k denote the security parameter. In the secret key generation phase, in addition to computing the secret trap-door pairs (f_0^{-1}, f_1^{-1}), (g_0^{-1}, g_1^{-1}) user A also picks a random function h in a poly-random collection F_k, and keeps h secret. (We assume that $k > b$.) During the signing process, A keeps a counter i to denote the number of times the signing algorithm has been invoked. To sign message m_i, A signs as before, except that (using m to denote the length of j):

- Instead of picking values r_j^f at random from D_f, he *computes* them as $r_j^f = h(0^{k-m}j)$.
- Instead of picking values r_j^g at random from D_g, he *computes* them as $r_j^g = h(1^{k-m}j)$.

We claim that the "memoryless" version of the signature scheme described above enjoys the same security properties as our original scheme. The proof (which we shall not give in detail) is based on the observation that if the memoryless scheme was vulnerable to an adaptive chosen-message attack, then it would be possible to efficiently distinguish pseudo-random functions from truly random functions.

A further improvement (due to Oded Goldreich [Go86]) removes even the necessity of remembering the number of previous signatures, by picking the index i for a message M as a random b-bit string. To make this work, the maximum number of signatures that can be produced by an instance of this scheme is limited to $2^{\sqrt{b}}$, so that it is extremely unlikely that two messages would have the same index chosen for them. The security proof can be modified to accomodate these changes. (Note that in the preprocessing step that builds an f-tree, we would now only build a portion of it consisting of $2^{\sqrt{b}}$ randomly chosen paths of length b.)

11. OPEN PROBLEMS

- It is an open question whether the RSA scheme is universally forgeable under an adaptive chosen-message attack.
- Can an encryption scheme be developed for which decryption is provably equivalent to factoring yet for which an adaptive chosen ciphertext attack is of no help to the enemy?

12. ACKNOWLEDGEMENTS

We are most grateful for Leonid Levin for his suggestion of how to use random functions in our scheme, in order to (almost) completely eliminate the need of storage in our signature scheme.

We are also very grateful to Oded Goldreich for many valuable suggestions concerning the presentation of these results, and for suggesting the speed-up described in section 10.3.

We are also thankful to Avi Wigderson for helpful suggestions on the presentation of this research.

Special thanks to an anonymous referee for his very careful reading of our paper and suggested improvements.

13. REFERENCES

[BGMR85] Ben-Or, M., O. Goldreich, S. Micali, and R.L. Rivest, "A Fair Protocol for Signing Contracts," *Proc. 12-th ICALP Conference* (Napflion, Greece, July 1985), 43–52.

[Bl82] Blum, M. "Coin Flipping by Telephone," *Proc. IEEE Spring COMPCOM* (1982), 133-137.

[Bl83] Blum, M. "How to Exchange (Secret) Keys" *ACM Trans. Comp. Sys.* **1** (1983), 175–193.

[BD85] Brickell, E., and J. DeLaurentis, "An Attack on a Signature Scheme Proposed by Okamoto and Shiraishi," Proc. CRYPTO 85 (Springer 1986).

[Ch82] Chaum, D. "Blind Signatures and Untraceable Payments," Advance in Cryptography – Proceedings of CRYPTO 82, (Edited by Chaum, D., R. Rivest, and A. Sherman), (Plenum Press, New York 1983).

[De82] Denning, D. CRYPTOGRAPHY AND DATA SECURITY, (Addison-Wesley, Reading, Mass., 1982).

[DH76] Diffie, W. and M. E. Hellman, "New Directions in Cryptography", *IEEE Trans. Info. Theory* **IT-22** (Nov. 1976), 644-654.

[EAKMM85] Estes, D., L. Adleman, K. Kompella, K. McCurley, and G. Miller, "Breaking the Ong-Schnorr-Shamir Signature Scheme for Quadratic Number Fields," *Proc. CRYPTO 85*, to appear.

[EG84] El-Gamal, T., "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms", *Proceedings of Crypto 84* (Springer 1985), 10–18.

[EGL82] Even, S., O. Goldreich, and A. Lempel, "A Randomized Protocol for Signing Contracts", *Advances in Cryptology – Proceedings of Crypto 82*, (Plenum Press, New York, 1983), 205-210.

[GGM84] Goldreich, O., Goldwasser, S., and S. Micali, "How to Contruct Random Functions," *Proc. 25th Annual IEEE Symposium on Foundations of Computer Science*, (Florida, November 1984.)

[Go86] Goldreich, Oded, "Two Remarks Concerning the GMR Signature Scheme," (Manuscript in preparation).

[GK86] Goldwasser, S., and J. Kilian, "Almost All Primes Can be Quickly Certified," *Proc. 18-th ACM STOC Conference* (Berkely, 1986).

[GM82] Goldwasser, S., and S. Micali, "Probabilistic Encryption," *JCSS* **28** (April 1984), 270-299.

[GMR84] Goldwasswer, S., S. Micali, and R. L. Rivest, "A 'Paradoxical' Solution to the Signature Problem," *Proc. 25-th IEEE FOCS Conference* (Singer Island, 1984), 441-448.

[GMY83] Goldwasser, S., S. Micali, and A. Yao, "Strong Signature Schemes," *Proc. 15th Annual ACM Symposium on Theory of Computing*, (Boston Massachusetts, April 1983), 431-439.

[La79] Lamport, L. "Constructing Digital Signatures from a One-Way Function," SRI Intl. CSL-98. (Oct. 1979)

[**Li81**] Lieberherr, K. "Uniform Complexity and Digital Signatures," *Theoretical Computer Science* **16**,1 (Oct. 1981), 99-110.

[**LM78**] Lipton, S., and S. Matyas, "Making the Digital Signature Legal – and Safeguarded," *Data Communications* (Feb. 1978), 41-52.

[**Ma79**] Matyas, S. "Digital Signatures – An Overview," *Computer Networks* **3** (April 1979) 87-94.

[**MH78**] Merkle, R., and M. Hellman, "Hiding Information and Signatures in Trap-Door Knapsacks," *IEEE Trans. Infor. Theory* **IT-24** (Sept. 1978), 525-530.

[**Me79**] Merkle, Ralph "Secrecy, Authentication, and Public-Key Systems," Stanford Electrical Engineering Ph.D. Thesis ISL SEL 79-017.

[**MM82**] Meyer, C. and S. Matyas, CRYPTOGRAPHY: A NEW DIMENSION IN DATA SECURITY (Wiley, New York, 1982)

[**MGR85**] Micali, S., S. Goldwasser, and C. Rackoff, "The Knowledge Complexity of Interactive Proof Systems," Proc. 17th Annual ACM Symposium on Theory of Computing, (Providence, R.I., May 1985), 291-304.

[**OS85**] Okamoto, T., and A. Shiraishi, "A Fast Signature Scheme Based on Quadratic Inequalities," Proc. 1985 Symp. on Security and Privacy (Oakland, April 1985).

[**OSS84a**] Ong, H., C. Schnorr, and A. Shamir, "An Efficient Signature Scheme Based on Quadratic Equations," *Proc. 16th Annual ACM Symposium on Theory of Computing*, (Washington, D.C., April 1984), 208-217.

[**OSS84b**] Ong, H., C. Schnorr, and A. Shamir, "An Efficient Signature Scheme Based on Polynomial Equations," *Proc. CRYPTO 84* (Springer 1985), 37–46.

[**Po84**] Pollard, J. "How to Break The 'OSS' Signature Scheme", Private Communication (1984).

[**Ra78**] Rabin, Michael, "Digitalized Signatures," In FOUNDATIONS OF SECURE COMPUTATION, (Edited by R. A. DeMillo, D. Dobkin, A. Jones, and R. Lipton), (Academic Press, New York, 1978), 133-153.

[**Ra79**] Rabin, Michael. "Digitalized Signatures as Intractable as Factorization," MIT Laboratory for Computer Science Technical Report MIT/LCS/TR-212 (Jan. 1979).

[**Ra80**] Rabin, Michael. "Probabilistic Algorithms for Testing Primality," *J. Number Theory*, **12** (1980), 128-138.

[**RV83**] Reif, J. and L. Valiant, "A logarithmic time sort for linear size networks," *Proceedings 15th Annual ACM Symposium on Theory of Computing*, (Boston Massachusetts, April 1983), 10-16.

[**RSA78**] Rivest, R., A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Comm. of the ACM* (Feb. 1978), 120-126.

[**Sh78**] Shamir, A., "A Fast Signature Scheme," MIT Laboratory for Computer Science Technical Memo MIT/LCS/TM-107 (July 1978).

[**Sh82**] Shamir, A., "A Polynomial Time Algorithm for Breaking the Basic Merkle-Hellman Cryptosystem," *Proc. 23rd Annual IEEE FOCS Conference* (Nov. 1982), 145-152.

[**SS77**] Solovay, R., and V. Strassen, "A Fast Monte-Carlo Test for Primality," *SIAM J. Computing*, **6** (1977), 84-85.

[**Tu84**] Tulpan, Y., "Fast Cryptanalysis of a Fast Signature System," Master's Thesis in Applied Mathematics, Weizmann Institute. (1984)

[**Wi80**] Williams, H. C., "A Modification of the RSA Public-Key Cryptosystem," *IEEE Trans. Info. Theory* **IT-26** (Nov. 1980), 726-729.

[**Yu79**] Yuval, G., "How to Swindle Rabin," *Cryptologia* **3** (July 1979), 187-189.