How To Sign Given Any Trapdoor Function

(extended abstract)

Mihir Bellare Silvio Micali*

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

Abstract

We present a digital signature scheme based on trapdoor permutations. This scheme is secure against *existential forgery under adaptive chosen message attack.* The only previous scheme with the same level of security was based on factoring.

Although the main thrust of our work is the question of reduced assumptions, we believe that the scheme itself is of some independent interest. We mention improvements on the basic scheme which lead to a memoryless and more efficient version.

1 INTRODUCTION

In 1976 Diffie and Hellman proposed that modern cryptography be based on the notions of one-way functions (functions that are easy to evaluate but hard to invert) and trapdoor functions (functions that are easy to evaluate and hard to invert without the possession of an associated secret).

In a few years, though, it became clear that basing security solely on assumptions as general as the existence of one-way or trapdoor functions was indeed a great challenge. The first provably good solutions that were found for several cryptographic problems were based on simple complexity theoretic assumptions about the computational difficulty of particular problems such as integer factorization ([GM], [Y], [BlMi]). More recently, however, it was found that pseudo-random number generation is possible if and only if certain kinds of one-way functions exist ([Y],[Le],[GKL]). Similarly we now know that secure encryption is possible if and only if trapdoor predicates exist.

Thus Diffie and Hellman's original goal was realized for two of the major cryptographic primitives. Somewhat surprisingly, in sharp contrast with the progress made

* supported in part by NSF grant DCR-84-13577 and ARO grant DAALO3-86-K-0171

in encryption and pseudo-random number generation, digital signatures, the other fundamental cryptographic primitive, was not yet based on a general assumption. The first paper to address the issues of security in a sufficiently general way and provide a signature scheme with a proof of security was that of [GMY]. Their results and the underlying notions of security were further improved in [GMR]. But both their schemes were based on factoring. Actually, [GMR] base their scheme on the existence of *claw-free pairs*, an assumption weaker than factoring but stronger than the assumption that trapdoor functions exist.

Thus, not only did we not know whether digital signatures are available based on a general complexity theoretic assumption but, even worse, digital signatures were totally linked to a single candidate hard problem. This is particularly unsatisfactory as a great many protocols make use of digital signatures and thus the computational intractability of factoring becomes a bottleneck in the assumptions of many cryptographic protocols.

The contribution of this paper is to free digital signatures from the fortunes of a specific algebraic problem by establishing a truly general signature scheme. Namely, we prove the following

Main Theorem: Secure digital signature schemes exist if *any* trapdoor permutation exists.

Thus, we show once again the feasibility of Diffie and Hellman's original proposal. To appreciate the generality of our theorem, let us clarify what "secure" and "trapdoor" mean.

By secure we mean "non existentially forgeable under an adaptive chosen message attack" as defined by [GMR]. Informally, in an adaptive chosen message attack, a polynomial time enemy (who sees the public key) can choose any message he wants and request to have it signed. After seeing the desired signature, the enemy can choose another message to be signed; and so forth for a polynomial (in the security parameter) number of times. Not to be existentially forgeable means that, after the attack, the enemy will not be able to sign any new message; that is, he will not be able to produce the signature of any string for which he had not previously requested and obtained the signature.

We believe this to be the strongest natural notion of security. In essence, in a scheme secure in this sense, signing is not only hard, but remains hard even having a "teacher" for it. This is more than we may need in practice, where an enemy may be able perhaps to see a few message-signature pairs, but is not able to ask for signatures of messages of his choice!

By trapdoor function we mean a permutation f hard to invert (without knowledge of the secret) on a polynomial fraction of the k-bit strings (when f has security parameter k). Notice that this underlying trapdoor f may by itself be insecure against an adaptive chosen message attack; that is, after being given the value of f^{-1} at a few chosen inputs, one may be able to easily invert f on all inputs. Our scheme will work with such functions as well. In fact our construction has a strengthening effect, and the resulting signing algorithm will be more secure than the trapdoor function it uses.

We are indebted to [GMR] not only as a source of ideas for the present paper, but also for their development and exposition of the notions of signatures and security on which we model our own.

2 SIGNATURE SCHEMES AND THEIR SECURITY

In a digital signature scheme, each user A publishes a "public key" while keeping secret a "secret key". User A's signature for a message m is a value depending on m and his public and secret keys such that anyone can verify the validity of A's signature using A's public key. However, it is hard to forge A's signatures without knowledge of his secret key. Below we give a more precise outline of the constituents of a signature scheme and of our notion of security against adaptive chosen message attack. We follow [GMR] for these notions.

2.1 Components of a Signature Scheme

A digital signature scheme has the following components:

- A *security parameter* k which is chosen by the user when he creates his public and secret keys and which determines overall security, the length and number of messages, and the running time of the signing algorithm.

- A *message space* which is the set of messages to which the signature algorithm may be applied. We assume all messages are binary strings, and to facilitate our exposition and proofs we assume that the message space is $\mathcal{M}_k = \{0,1\}^k$, the set of all k-bit strings, when the security parameter is k.

- A polynomial S_B called the *signature bound.* The value $S_B(k)$ represents a bound on the number of messages that can be signed when the security parameter is k.

- A probabilistic polynomial time *key generation algorithm* KG which can be used by any user to produce, on input 1^k, a pair (PK, SK) of matching public and secret keys.

- A probabilistic polynomial time *signing algorithm* $\mathcal{S}$ which given a message m and a pair (PK, SK) of matching public and secret keys, produces a signature of m with respect to PK. $\mathcal{S}$ might also have as input the signatures of all previous messages it has signed relative to PK.

- A polynomial time *verification algorithm* $\mathcal{V}$ which given S, m, and PK tests whether S is a valid signature for the message m with respect to the public key PK.

Note that the key generation algorithm must be randomized to prevent a forger from re-running it to obtain a signer's secret key. The signing algorithm need not be randomized, but ours is; a message may have many different signatures depending on the random choices of the signer.

2.2 Security against Adaptive Chosen Message Attacks

Of the various kinds of attacks that can be mounted against a signature scheme by a forger, the most general is an *adaptive chosen method attack*. Here a forger uses the signer A to obtain sample signatures of messages of his choice. He requests message signatures, with his requests depending not only on A's public key but on the signatures returned by A in response to the forger's previous requests. From the knowledge so gathered he attempts forgery.

The most general kind of forgery is the successful signing, relative to A's public key, of *any* message m. This is called an *existential forgery*. (Note that forgery of course only denotes the creation of a *new* signature; it is no forgery to obtain a valid signature from A and then claim to have "forged" it). The security we require of our scheme is that existential forgery under an adaptive chosen message attack be infeasible with very high probability. For qualifications and a more precise expression of these notions we resort to a complexity theoretic framework.

A *forger* is a probabilistic polynomial time algorithm $\mathcal{F}$ which on input a public key PK with security parameter k

- engages in a conversation with the legal signer S, requesting and receiving signatures for messages of his choice, for a total number of messages bounded by a polynomial in k (the adaptive chosen message attack),
- then outputs S purporting to be a signature with respect to PK of a new message m (an attempt at existential forgery).

We say $\mathcal{F}$ is *successful* if the signature it creates is a genuine (i.e. $\mathcal{V}(S, m, PK) = \mathtt{true}$) forgery. We say that a signature scheme is *Q-forgable* (Q a polynomial) if there exists a forger $\mathcal{F}$ who, for infinitely many k, succeeds with probability more than $\frac{1}{Q(k)}$ on input a public key with security parameter k. The probability here is over the choice of the public key, which is chosen according to the distribution generated by KG, and over the coin tosses of $\mathcal{F}$ and S.

The security property we are interested in consists of not being Q-forgable for any polynomial Q.

3 TRAPDOOR PERMUTATIONS

We propose here a relatively simple complexity theoretic definition of trapdoor permutations which nevertheless captures all the known candidates for trapdoor permutations.

Definition 3.1 A triplet (G, E, I) of probabalistic polynomial time algorithms is a *trapdoor permutation generator* if on input 1^k the algorithm G outputs a pair of k bit strings (x, y) such that

(1) The algorithms $E(x, \cdot)$ and $I(y, \cdot)$ define permutations of $\{0,1\}^k$ which are inverses of each other: $I(y, E(x, z)) = z$ and $E(x, I(y, z)) = z$ for all $z \in \{0,1\}^k$.

(2) For all probabilistic polynomial time (adversary) algorithms $A(.,.,.)$ and for all c and sufficiently large k,
$$Pr[E(x, A(1^k, x, z)) = z] < k^{-c}$$
when z is chosen at random from $\{0,1\}^k$ and the pair (x, y) is obtained by running the generator on input 1^k (the probability is over the random choice of z and the coin tosses of G and A).

The algorithms G, E and I are called the *generating, evaluating* and *inverting* algorithms respectively.

Definition 3.2 A function f is a *trapdoor permutation* with *security parameter* k if there is a trapdoor permutation generator (G, E, I) such that $f = E(x, \cdot)$ for a pair of strings (x, y) obtained by running G on input 1^k.

Notice that as defined above, a trapdoor permutation with security parameter k has domain all of $\{0,1\}^k$. This is not the case with known candidates such as *RSA* ([RSA]) or the trapdoor permutations of [BBS] where the domain is a subset of $\{0,1\}^k$. Also notice that the probability of inversion that we require in part (2) of the definition (k^{-c}) looks very low. Both of these, though, are not restrictions; all the known candidates can be fit into our scenario by using a cross product construction as in [Y]. This works as follows.

Given a trapdoor permutation f on a subset D of $\{0,1\}^k$ such that $|D| \geq 2^k \cdot k^{-d}$ for some d and f is hard to invert on all but a polynomial fraction of D, extend f to $\{0,1\}^k$ by defining it to be the identity function on $\{0,1\}^k - D$. This yields a permutation $\bar{f}$ on $\{0,1\}^k$. Define a function F on
$$\underbrace{\{0,1\}^k \times \ldots \times \{0,1\}^k}_{k^{d+2}}$$
by $F(x_1, \ldots, x_{k^{d+2}}) = (\bar{f}(x_1), \ldots, \bar{f}(x_{k^{d+2}}))$. F is a permutation and Yao shows that it satisfies part (2) of Definition 3.1 given our assumptions about the original f.

4 AN OVERVIEW OF THE SCHEME

We present here an overview of the scheme and a sketch of the proof of security; the succeeding sections gives a more complete description and proof. In this section, as well as in the complete scheme we describe later, we disregard efficiency completely for the sake of simplicity.

4.1 Background

In [La] Lamport suggested the following method for signing a single bit: make public f and a pair of points x^0 and x^1 and keep secret f^{-1}. The signature of a bit $b \in \{0,1\}$ is then $f^{-1}(x^b)$. The drawback of this method is that the number of bits that can be signed is limited to the number of pairs of points that are placed in the public key. Our scheme can be considered an extension of this type of scheme in that it removes the restriction on the number of bits that can be signed while using a similar basic format for signing a single bit. We do this by regenerating some of the public key information every time we sign a bit. [GMR] too uses the idea of regenerating some part of the information in the public key, but with a different, non Lamport like underlying signing method. Merkle ([M]) presents another way of extending the Lamport format; his more pragmatically oriented scheme, though, is not concerned with proofs of security.

In the scheme described below, and then in more detail in §5, we reverse the roles of functions and points in the Lamport format with respect to signing a single bit, and then sign new points as needed. (A dual and equivalent scheme consists of directly using the Lamport format but signing new *functions* instead; this was in fact the way our scheme was presented in [BeMi]).

4.2 The Signature Scheme

A user's public key in our scheme is of the form

$$PK = (f_{0,0}, f_{0,1}, \ldots, f_{k,0}, f_{k,1}, \alpha)$$

where the $f_{i,j}$ are trapdoor permutations with security parameter k and α is a random k bit string (we refer to k bit strings equivalently as *points* or *seeds*). His secret key is the trapdoor information $f_{i,j}^{-1}$. A message is signed bit by bit. The first bit b_1 is signed by sending $f_{0,b_1}^{-1}(\alpha)$ and *a signature of a new seed* α_1. The signature of the k bit string α_1 consists of sending, for each $i = 1, \ldots, k$, either $f_{i,0}^{-1}(\alpha)$ or $f_{i,1}^{-1}(\alpha)$ depending on whether the i-th bit of α_1 was a 0 or a 1.

At this point not only has the bit b_1 been signed, but the public key has been "recreated". That is, another bit can now be signed in the same manner with α_1 playing the role of α above. This process can be continued to sign a polynomial in k number of bits. The signature of a message is thus built on a chain of seeds in which each element of the chain is used to sign its successor.

4.3 Why is this Secure?

Suppose $\mathcal{F}$ is a forger (as described in §2.2). We derive a contradiction by showing that the existence of $\mathcal{F}$ implies the existence of an algorithm A which inverts the underlying trapdoor permutations with high probability.

Given a trapdoor permutation g with security parameter k and a k bit string z, the algorithm A must use the forger to find $g^{-1}(z)$. A's strategy will be to build a suitable public key and then run $\mathcal{F}$ and attempt to sign the messages requested by $\mathcal{F}$. From $\mathcal{F}$'s forged signature will come the information required to invert g.

The public key

$$PK = (f_{0,0}, f_{0,1}, \ldots, f_{k,0}, f_{k,1}, \alpha)$$

that A creates has $f_{n,c} = g$ for some n and c. All the other functions are obtained by running the generator, so A knows their inverses. In the course of signing A will use a list of seeds of the form $g(\alpha_l)$, except for some one stage at which it will use as seed the given point z. So A knows how to invert all the $f_{i,j}$ at all the seeds with the single exception of not knowing $f_{n,c}^{-1}(z)$. At this point, it is possible that A will not be able to sign a message that $\mathcal{F}$ requests. Specifically, A will not be able to sign a message m if computing the signature would require knowledge of $g^{-1}(z)$. But this is the only possible block in A's signing process, and it will happen with probability only $1/2$. So A succeeds in responding to all $\mathcal{F}$'s requests with probability $1/2$.

By assumption $\mathcal{F}$ will now return the signature of a message not signed previousley by A. The placement of the original function g in the public key, as well as the placement of z in the list of seeds, are unknown to $\mathcal{F}$ (more precisely, the probability distribution of real signatures and A's signatures are the same). With some sufficiently high probability, the signature of the new message will include the value of $g^{-1}(z)$ which A can output and halt.

5 THE SCHEME AND PROOF OF SECURITY

5.1 Preliminary Notation and Definitions

The i-th bit of a binary string x is denoted $(x)_i$ while its length is denoted $|x|$.

If $a = (a_1, \ldots, a_i)$ and $b = (b_1, \ldots, b_j)$ are sequences then $a*b$ denotes the sequence $(a_1, \ldots, a_i, b_1, \ldots, b_j)$. If $a = (a_1, \ldots, a_i)$ is a sequence and $j \leq i$ then $(a_1, \ldots, a_j)$ is called an initial segment of a.

We recall [GMR]'s notation and conventions for probabalistic algorithms. If A is a probabalistic algorithm then $A(x, y, \ldots)$ denotes the probability space which assigns to the string σ the probability that A, on input $x, y, \ldots$, outputs σ. We denote by $[A(x, y, \ldots)]$ the set of elements of $A(x, y, \ldots)$ which have non-zero probability. We denote by $x \leftarrow A(x, y, \ldots)$ the algorithm which assigns to x a value selected according to the probability distribution $A(x, y, \ldots)$. If S is a finite set we write $x \leftarrow S()$ for the algorithm which assigns to x a value selected from S uniformly at random. The notation

$$\boldsymbol{P}(p(x, y, \ldots) \; : \; x \leftarrow S; y \leftarrow T; \ldots)$$

denotes the probability that the predicate $p(x, y, \ldots)$ is true after the (ordered) execution of the algorithms $x \leftarrow S$, $y \leftarrow T$, etc. As an example of this notation, part (2) of Definition 3.1 would be written as

$$\boldsymbol{P}(E(x, u) = z \; : \; (x, y) \leftarrow G(1^k); z \leftarrow \{0,1\}^k; u \leftarrow A(1^k, x, z)) < k^{-c}.$$

We let PPT denote the set of probabalistic polynomial time algorithms. We assume that a natural encoding of these algorithms as binary strings is used.

For the remainder of this section we fix a trapdoor permutation generator (G, E, I).

With some abuse of language we will often call x a function and identify it with $E(x, \cdot)$. In the scheme we now proceed to describe we will desregard efficiency completely in order to simplify the proof of security.

5.2 Building Blocks for Signing

The signing algorithm makes use of many structures. This section describes the basic building blocks that are put together to build signatures.

Let $(x_i^j, y_i^j) \in [G(1^k)]$ for $i = 0, \ldots, k$ and $j = 0, 1$, and let $\vec{x} = (x_0^0, x_0^1, \ldots, x_k^0, x_k^1)$, $\vec{y} = (y_0^0, y_0^1, \ldots, y_k^0, y_k^1)$. Let $\alpha, \alpha' \in \{0,1\}^k$.

Definition 5.1 A *seed authenticator* $\langle \alpha'; \alpha \rangle_{\vec{x}}$ is a tuple of strings $(\alpha', \alpha, z_1, \ldots, z_k)$ for which

$$E(x_i^{(\alpha)_i}, z_i) = \alpha' ,$$

for all $i = 1, \ldots, k$.

Definition 5.2 A *bit authenticator* $\langle \alpha'; b \rangle_{\vec{x}}$ is a tuple of strings (α', b, z) such that $b \in \{0,1\}$ and $E(x_0^b, z) = \alpha'$.

Definition 5.3 An *authenticator* $\langle \alpha'; c \rangle_{\vec{x}}$ is either a seed authenticator or a bit authenticator. In the authenticator $\langle \alpha'; c \rangle_{\vec{x}}$, α' is called the *root* of the authenticator, c is called the *child* of the authenticator, and $\vec{x}$ is called the *source* of the authenticator.

Given $\vec{x}$ and a tuple purporting to be an authenticator $\langle \alpha'; c \rangle_{\vec{x}}$, it is easy for anyone to check that it is indeed one. However given α', c, and $\vec{x}$ it is difficult to create an authenticator $\langle \alpha'; c \rangle_{\vec{x}}$ without the knowledge of $\vec{y}$.

Definition 5.4 A sequence $F = (F^1, \ldots, F^p)$ of seed authenticators is a *spine starting at* α' if

- α' is the root of F^1.
- for $i = 1, \ldots, p-1$, the root of F^{i+1} is the child of F^i.

Definition 5.5 A sequence $B = (B^1, \ldots, B^q)$ of bit authenticators is *s-attached* to the spine $F = (F^1, \ldots, F^p)$ if the root of B^i is equal to the child of F^{s+i-1} for $i = 1, \ldots, q$. A sequence of bit authenticators $B = (B^1, \ldots, B^q)$ is *attached* to the spine $F = (F^1, \ldots, F^p)$ if it is s-attached for some s.

5.3 Generating Keys

The key generation algorithm KG does the following on input 1^k:

(1) Run G a total of $2k + 2$ times on input 1^k to get a list of pairs (x_i^j, y_i^j) $(i = 0, \ldots, k,\ j = 0, 1)$.

(2) Select a random k-bit seed $\alpha \in \{0,1\}^k$.

(3) Output the public key $PK = (1^k, \vec{x}, \alpha, S_B)$ where $\vec{x} = (x_0^0, x_0^1, \ldots, x_k^0, x_k^1)$ and S_B is the signature bound.

(4) Output the secret key $SK = \vec{y} = (y_0^0, y_0^1, \ldots, y_k^0, y_k^1)$.

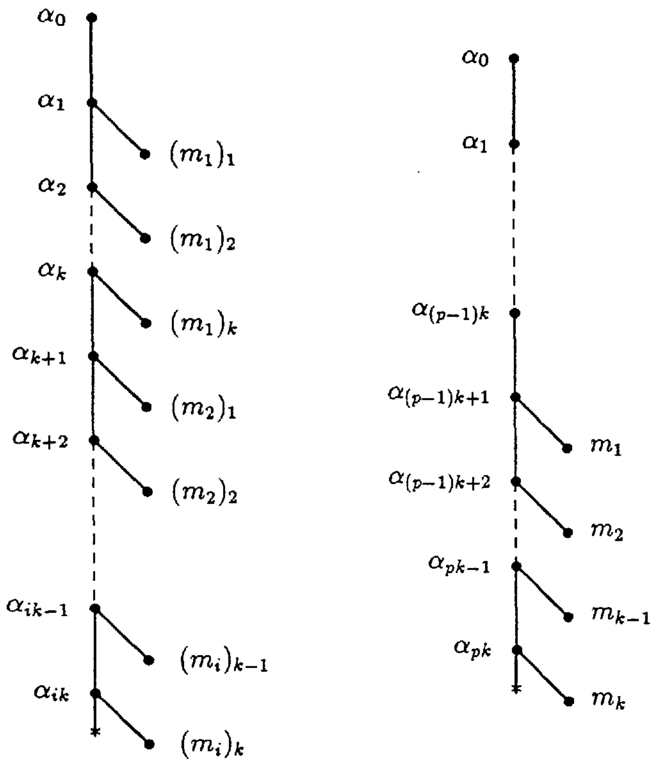

Figure 1: A signature corpus (left), and a signature of a message m (right)

5.4 What is a Signature?

Definition 5.6 A *signature* of a message $m \in \mathcal{M}_k$ with respect to a public key $PK = (1^k, \vec{x}, \alpha, S_B)$ is a triple (F, B, m) where $F = (F^1, \ldots, F^{pk})$ ($p \geq 1$) is a spine and $B = (B^1, \ldots, B^k)$ is a sequence of bit authenticators such that

- B is $((p-1)k+1)$-attached to F.
- F starts at α.
- For all $i = 1, \ldots, k$ the child of B^i is $(m)_i$.
- The common source of all the authenticators is $\vec{x}$.

Figure 2 shows a schema of a signature for a message m with respect to a public key $(1^k, \vec{x}, \alpha_0, S_B)$; here $F^i = \langle \alpha_{i-1}; \alpha_i \rangle_{\vec{x}}$ $(i = 1, \ldots, pk)$ and $B^i = \langle \alpha_{(p-1)k+i}; (m)_i \rangle_{\vec{x}}$ $(i = 1, \ldots, k)$.

5.5 The Signing Algorithm and Signature Corpus

Let $PK = (1^k, \vec{x}, \alpha_0, S_B)$ and $SK = \vec{y}$ be a pair of public and secret keys. We presume that the signing procedure $\mathcal{S}$ is initialized with the values of PK and SK and has already signed messages $m_1, \ldots, m_{i-1}$ and kept track of the signatures $S_1 =$

$(F_1, B_1, m_1), \ldots, S_{i-1} = (F_{i-1}, B_{i-1}, m_{i-1})$ of these messages. We let F_0 be the empty sequence. To compute a signature $S_i = (F_i, B_i, m_i)$ for m_i, where $i \leq S_B(k)$ and $m_i \in \mathcal{M}_k$, $\mathcal{S}$ performs the following steps:

(1) Set $l = (i-1)k$, and select k seeds $\alpha_{l+1}, \ldots, a_{l+k} \in \{0,1\}^k$ at random.

(2) Form the seed authenticators
$$F^j = \langle \alpha_{j-1} ; \alpha_j \rangle_{\vec{x}} \,,$$
for $j = l+1, \ldots, l+k$, and let F be the spine $(F^{l+1}, \ldots, F^{l+k})$.

(3) Form the bit authenticators
$$B^j = \langle \alpha_{l+j} \, ; (m_i)_j \rangle_{\vec{x}} \,,$$
for $j = 1, \ldots, k$, and let $B_i = (B^1, \ldots, B^k)$.

(4) Let $F_i = F_{i-1} * F$ and output $S_i = (F_i, B_i, m_i)$ as the signature of m_i.

Figure 1 shows a schema of the data structure constructed by the signing procedure as described above. This structure will be called a signature corpus below.

Definition 5.7 Let
$$(F_1, B_1, m_1), \ldots, (F_i, B_i, m_i)$$
be a sequence of the first i signatures output by our signing algorithm $\mathcal{S}$, for some $i > 0$. Let $F = F_i$ and $B = B_1 * \ldots * B_i$. We call *signature corpus* the triple $C = (F, B, (m_1, \ldots, m_i))$.

Note that a signature corpus (F, B, M) is a spine $F = (F^1, \ldots, F^p)$ to which is 1-attached the sequence of bit carrying items $B = (B^1, \ldots, B^p)$.

Definition 5.8 Let $Z = (F, B, M)$ be either a single signature or a signature corpus, relative to a public key $PK = (1^k, \vec{x}, \alpha_0, S_B)$, where $F = (F^1, \ldots, F^p)$ and $B = (B^1, \ldots, B^q)$. Then

(1) $F(Z)$ denotes F, the spine of Z, and $B(Z)$ denotes B, the sequence of bit authenticators of Z. The authenticators in F are called the seed authenticators of Z and the authenticators in B are called the bit authenticators of Z.

(2) The set of authenticators of Z is $A(Z) = \{F^1, \ldots, F^p\} \cup \{B^1, \ldots, B^q\}$.

(3) The chain of seeds of Z, denoted $P(Z)$, is the sequence of seeds which form the roots and children of the seed authenticators of F. That is, $P(Z) = (\alpha_0, \alpha_1, \ldots, \alpha_p)$, where α_i is the child of F^i for all $i = 1, \ldots, p$.

(4) The set of roots of Z, denoted $R(Z)$, is the set of roots of the seed authenticators of Z.

(5) The tuple M of messages signed by Z is denoted $M(Z)$. (If Z is the signature of a single message m, we just let $M(Z) = m$).

5.6 The Verification Algorithm

Given a public key PK and something purporting to be a signature of a message m with respect to PK, it is easy to check whether this is indeed the case. It is easy to see that checking whether a given object really has the form of definition 5.6 only requires knowledge of the public key.

5.7 Extracting Information From a Forgery

As indicated in the overview of §4.3, forgery must eventually be used to extract information about the inversion of a trapdoor function. The preliminary definitions and lemmas here are devoted to charecterizing the structure of a forgery relative to a given corpus.

Lemma 5.1 Let C be a signature corpus relative to a public key $PK = (1^k, \vec{x}, \alpha, S_B)$ and let S be a signature, relative to the same public key, of a message m not in $M(C)$. Then there is an α' in $P(C)$ such that one of the following holds:

(1) There is a pair of seed authenticators, $\langle\alpha'; h_1\rangle_{\vec{x}}$ in $F(C)$, and $\langle\alpha'; h_2\rangle_{\vec{x}}$ in $F(S)$, such that $h_1 \neq h_2$.

(2) α' is not in $R(C)$ (i.e. α' is the child of the last authenticator in the spine) and there is a seed authenticator $\langle\alpha'; h\rangle_{\vec{x}}$ in $F(S)$.

(3) There is a pair of bit authenticators, $\langle\alpha'; b_1\rangle_{\vec{x}}$ in $B(C)$, and $\langle\alpha'; b_2\rangle_{\vec{x}}$ in $B(S)$, such that $b_1 \neq b_2$.

Proof: Suppose neither (1) nor (2) holds. Since $F(S)$ and $F(C)$ both start at α, $F(S)$ must be an initial segment of $F(C)$. Thus $P(S)$ is an initial segment of $P(C)$. Since $B(S)$ is attached to $F(S)$, the roots of all the bit authenticators of S are in $P(S)$ hence in $P(C)$. So if $P(C) = (\alpha_0, \ldots, \alpha_{pk})$ then there is some i such that $\langle\alpha_{(i-1)k+j}; (m_i)_j\rangle_{\vec{x}} \in B(C)$ and $\langle\alpha_{(i-1)k+j}; (M(S))_j\rangle_{\vec{x}} \in B(S)$ for all $j = 1, \ldots, k$, where $m_i \in \mathcal{M}_k$ is the i-th message in the corpus. But $M(S)$ is not in $M(C)$, so there is some j such that $(M(S))_j \neq (m_i)_j$. Let $b_1 = (m_i)_j$, $b_2 = (M(S))_j$, and $\alpha' = \alpha_{(i-1)k+j}$. Then $\langle\alpha'; b_2\rangle_{\vec{x}} \in B(S)$ and $\langle\alpha'; b_1\rangle_{\vec{x}} \in B(C)$ are the desired bit authenticators which give us part (3) of the lemma. □

Let $PK = (1^k, \vec{x}, \alpha, S_B)$ be a public key, where $\vec{x} = (x_0^0, x_0^1, \ldots, x_k^0, x_k^1)$, and let C be a signature corpus relative to PK. We introduce the notion of a pair (α', x_i^j) being *unused* in C, where α' is in $P(C)$. Informally, we would like to say that (α', x_i^j) is unused if the authenticators in the corpus C do not contain $E(x_i^j, \cdot)^{-1}(\alpha')$. That is, the inversion of $E(x_i^j, \cdot)$ at α' was not required in the signing process. For technical reasons however, the formal definition that we use is rather to say that the inversion of $E(x_i^{1-j}, \cdot)$ *was* required in the signing process. Boundary conditions (being at the end of the spine) complicate things a little further.

Definition 5.9 Let PK, C be as above. We say that (α', x_i^j) is *unused* in C if α' is in $P(C)$ and one of the following holds:

(1) There is a seed authenticator $\langle\alpha'; h\rangle_{\vec{x}}$ in $A(C)$ with $(h)_i \neq j$.

(2) $i \neq 0$ and α' is not in $R(C)$. (So α' is at the tail end of the spine $F(C)$).

(3) $i = 0$ and there is a bit authenticator $\langle\alpha'; b\rangle_{\vec{x}}$ in $A(C)$ with $b \neq j$.

With PK, C as above, let S be the signature of a message m not in $M(C)$, relative to PK. We show that this signature could not have been created without inverting $E(x_i^j, \cdot)$ at α' where (α', x_i^j) was some unused pair in the corpus C.

Lemma 5.2 There is a polynomial time algorithm which takes as input PK, C, and S as described above, and outputs a triple of the form (α', x_i^j, u) such that the pair (α', x_i^j) was unused in C and $E(x_i^j, u) = \alpha'$.

Proof: Let α' be the seed of Lemma 5.1. The proof breaks down into the cases provided by Lemma 5.1, and we number the cases below accordingly. Note that given C and S it is possible for an algorithm to determine which of the cases of Lemma 5.1 applies.

(1) Since $h_1 \neq h_2$ we can find an i such that $(h_1)_i \neq (h_2)_i$. Set $j = (h_2)_i$. The authenticator $\langle\alpha'; h_2\rangle_{\vec{x}}$ provides us with the value $E(x_i^j, \cdot)^{-1}(\alpha')$, and by the first part of Definition 5.9 the pair (α', x_i^j) is unused in C.

(2) Set i to any value between 1 and k and set $j = (h)_i$. The authenticator $\langle\alpha'; h\rangle_{\vec{x}}$ provides us with the value $E(x_i^j, \cdot)^{-1}(\alpha')$, and the second part of Definition 5.9 says that (α', x_i^j) is unused in C.

(3) Set $i = 0$ and $j = b_2$. The authenticator $\langle\alpha'; b_2\rangle_{\vec{x}}$ provides us with the value $E(x_i^j, \cdot)^{-1}(\alpha')$ and the last part of Definition 5.9 says that (α', x_i^j) is unused in C. $\square$

5.8 Proof of Security

We are finally ready to prove

Theorem 5.1 Under the assumption that (G, E, I) is a trapdoor permutation generator the above signature scheme is not even Q-forgable (see §2.2), for all polynomials Q and all sufficiently large k.

The proof of the theorem is by contradiction. Assume the existence of a polynomial Q, an infinite set $\overline{K}$, and a forger $\mathcal{F}(\cdot)$ such that for all $k \in \overline{K}$, $\mathcal{F}$ is succesful in forging with probability $\geq \frac{1}{Q(k)}$ on input a public key chosen according to the distribution induced by KG. Our goal is to construct an algorithm $A(\cdot,\cdot,\cdot) \in PPT$ which on input $1^k, x, z$ uses $\mathcal{F}$ to find $E(x, \cdot)^{-1}(z)$.

A operates as follows on input $1^k, x, z$:

(1) Let $n \leftarrow \{0, \ldots, k\}()$, $c \leftarrow \{0, 1\}()$, and $t \leftarrow \{0, \ldots, kS_B(k)\}()$.

(2) Run G a total of $2k + 1$ times on input 1^k to get (x_i^j, y_i^j) for $i = 0, \ldots, k$, $j = 0, 1$, $(i, j) \neq (n, c)$. Let $x_n^c = x$, and let $\vec{x} = (x_0^0, x_0^1, \ldots, x_k^0, x_k^1)$.

(3) Pick $kS_B(k)$ random k bit strings $\beta_0, \ldots, \beta_{t-1}, \beta_{t+1}, \ldots, \beta_{kS_B(k)}$, and then create the seeds

$$\alpha_l = \begin{cases} z & \text{if } l = t \\ E(x, \beta_l) & \text{otherwise.} \end{cases}$$

Let P be the sequence $(\alpha_0, \alpha_1, \ldots, \alpha_{kS_B(k)})$.

(4) Let $PK = (1^k, \vec{x}, \alpha_0, S_B)$.

(5) Invoke $\mathcal{F}$ on the public key PK, and attempt to sign the requested messages in the same manner as the signing procedure $\mathcal{S}$, but using the already generated seeds from P where $\mathcal{S}$ would pick random new seeds. The inverses of all but one of the functions in $\vec{x}$ are known, and, for that function x_n^c, the value $E(x_n^c, \cdot)^{-1}(\alpha_l) = \beta_l$ is known for all values $l \neq t$. If either $(\alpha_{t+1})_n = c$, or $n = 0$ and the sequence of requested messages has c in the t-th position, it will not be possible to sign. Output $\emptyset$ and halt in this case. If all $\mathcal{F}$'s requested messages are succesfully signed, let C be the corpus of these signatures.

(6) If $\mathcal{F}$ does not now output a signature of a message not in $M(C)$, output $\emptyset$ and halt. Otherwise, invoke the algorithm of Lemma 5.2 on input PK, C, and the signature S output by $\mathcal{F}$. This algorithm outputs a tuple (α', x_i^j, u). Now output u and halt.

We consider the distribution of A's output when its inputs are chosen at random; that is, we consider the result of executing

$$(x, y) \leftarrow G(1^k); z \leftarrow \{0,1\}^k(); u \leftarrow A(1^k, x, z).$$

Lemma 5.3 The public key PK created in step 4 has the same distribution as that induced on public keys by the key generation algorithm KG.

Proof: The functions x_i^j of step 2 were obtained by running G, as was x, so $\vec{x}$ has the right distribution. The β_l were chosen at random in step 3. Since $E(x, \cdot)$ is a permutation, the seeds α_l are also randomly distributed. Since α_0 is either one of these or the randomly chosen z, it is randomly distributed. So PK has the same distribution as generated by KG (§5.3). $\square$

Lemma 5.4

(1) The distribution of signatures generated by the conversation between $\mathcal{F}$ and A is, at every stage in the conversation, the same as the distribution that would be generated in a conversation between $\mathcal{F}$ and the legal signer $\mathcal{S}$.

(2) With probability $\geq \frac{1}{2}$ all of $\mathcal{F}$'s requests are succesfully signed.

Proof: As noted above, the public key has the right distribution. Now the steps used by A to sign are exactly those of the signing algorithm $\mathcal{S}$, with the one exception noted in step 5 of the description of A. The signatures received by $\mathcal{F}$ upto this crucial point have the same distribution as the legal signer would have generated. Upto this point then, $\mathcal{F}$ sees no anomaly. Now at the next step A must invert either $E(x_n^0, \cdot)$ or $E(x_n^1, \cdot)$ at α_t. Since c was chosen at random, we can conclude that this stage is passed with probability $\frac{1}{2}$. Moreover, this and future signatures are still with the right distribution. Both parts of the lemma are thus verified. □

Suppose all $\mathcal{F}$'s requests are signed. By the preceding lemma, the corpus generated has the same distribution as would have been generated with the legal signer. By assumption we know $\mathcal{F}$ forges with probability $\frac{1}{Q(k)}$ on this distribution. Since the signing was accomplished with probability $\geq \frac{1}{2}$ we obtain a forgery S with probability

$$\geq \frac{1}{2Q(k)} .$$

The next step is to show that the u output by A is equal to $E(x,\cdot)^{-1}(z)$ with sufficiently high probability.

Note that $P(C)$ is an initial segment of the sequence P. If the requested messages added together to a length of more than t bits, then z is in $P(C)$. The signing process is accomplished only if inverting $E(x,\cdot) = E(x_n^c,\cdot)$ at z is avoided, so if z is in $P(C)$ then (x,z) is unused in C. We state this as a lemma.

Lemma 5.5 If A does succeed in signing all of $\mathcal{F}$'s requests, and if z is in $P(C)$, then (z,x) is unused in C.

Proof: If z is the last seed in the sequence $P(C)$ and $n > 0$ then we have case (2) of Definition 5.9. Otherwise, since the signing was accomplished, either (1) or (3) must hold. □

By Lemma 5.2, $u = E(x_i^j,\cdot)^{-1}(\alpha')$ for some pair (α', x_i^j) unused in C. We would like the pair to actually be (z,x), for then $u = E(x,\cdot)^{-1}(z)$. The randomization of the n and t parameters (step 1) serves to capture this event with probability at least

$$\frac{1}{(1+k)(1+kS_B(k))} .$$

We conclude that for all $k \in \overline{K}$,

$$P(E(x,u) = z \;:\; (x,y) \leftarrow G(1^k); z \leftarrow \{0,1\}^k(); u \leftarrow A(1^k,x,z))$$
$$\geq \frac{1}{2Q(k)(1+k)(1+kS_B(k))} ,$$

contradicting the fact that G is a trapdoor permutation generator. This completes the proof of Theorem 5.1.

6 VARIATIONS AND IMPROVEMENTS

The signatures produced by the signing algorithm of the previous section are far from compact: signatures with respect to a public key $PK = (1^k, \vec{x}, \alpha, S_B)$ could reach lengths of $O(kS_B(k))$. We describe briefly here how tree structures in the style of [GMR] could replace the linear structures of the above scheme to produce signatures of length $O(k \log S_B(k))$. The size of signatures in the modified scheme will not only be smaller but will be independent of the signatures of previous messages. The modified scheme retains the security properties of the original one.

The public key now contains $2k+1$ pairs of randomly chosen trapdoor functions of security parameter k together with, as before, a single seed. Each seed is used to sign two others, which become its right and left children in the tree; the first k pairs of the above functions are used to sign the left child, and the second k pairs to sign the right child. The tree is grown to height $k \log S_B(k)$. Each leaf can then be used as the root of a linear chain of length k which signs a single message. The proof of security needs little change for the modified scheme, and details are left to the final paper.

The assumption that messages are always of length equal to the security parameter can be removed: to sign messages of arbitrary length it suffices to first encode them with a *subsequence free encoding*. This is an encoding which guarantees that no string is a substring of the concatenation of any number of other strings, and such encodings are easy to construct.

Further, the scheme, in its tree version, can be made *memoryless* (as in the modifications of [Go] and [Gu] to the [GMR] scheme); the same ideas used by [GMR] (attributed to Levin), and extended in [Go], can be applied here. The main tool is the use of pseudo-random functions ([GGM]) whose existence is implied by our assumptions.

References

[BeMi] Bellare, M., and S. Micali, "How to Sign Given Any Trapdoor Function," *Proceedings of the 20th STOC,* ACM (1988), 32-42.

[BBS] Blum, L., M. Blum, and M. Shub, "A Simple Unpredictable Pseudo-Random Number Generator," *SIAM Journal on Computing*, Vol. 15, No. 2 (May 1986), 364-383.

[BlMi] Blum, M., and S. Micali, "How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits," *SIAM Journal on Computing*, Vol. 13, No. 4 (November 1984), 850-864.

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

[Go] Goldreich, O., "Two Remarks Concerning the GMR Signature Scheme," MIT Laboratory for Computer Science Technical Report 715, (September 1986).

[GKL] Goldreich, O., M. Luby, and H. Krawczyk, "On the Existence of Pseudo-random Generators," *CRYPTO 88*.

[GGM] Goldreich, O., S. Goldwasser, and S. Micali, "How To Construct Random Functions," *Journal of the Association for Computing Machinery*, Vol. 33, No. 4 (October 1986), 792-807.

[GM] Goldwasser, S., and S. Micali, "Probabalistic Encryption," *Journal of Computer and System Sciences* 28 (April 1984), 270-299.

[GMR] Goldwasser, S., S. Micali and R. Rivest, "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks," *SIAM Journal on Computing*, vol. 17, No. 2, (April 1988), 281-308.

[GMY] Goldwasser, S., S. Micali, and A. Yao, "Strong Signature Schemes," *Proceedings of the 15th STOC,* ACM (1983), 431-439.

[Gu] Guillou, L., "A Zero-Knowledge Evolution of the Paradoxical GMR Signature Scheme", manuscript (February 1988).

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

[Le] Levin, L., "One Way Functions and Pseudo Random Generators," *Proceedings of the 17th STOC,* ACM (1985), 363-365.

[M] Merkle, R., "A Digital Signature Based on a Conventional Encryption Function," *Advances in Cryptology - CRYPTO 87 (Lecture Notes in Computer Science, 293)*, Springer-Verlag, 1987.

[RSA] Rivest, R., A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM* (Feb 78), 120-26.

[Y] Yao, A. C., "Theory and Applications of Trapdoor Functions," *Proceedings of the 23rd FOCS*, IEEE (1982) 80-91.