Appears in *Journal of the ACM*, Vol. 39, No. 1, January 1992, pp. 214–233. Prelimanary version in Proceedings *of the* 20th *Annual Symposium on Theory of Computing*, ACM, 1988.

How to Sign Given Any Trapdoor Permutation

Mihir Bellare* Silvio Micali†

January 1992

Abstract

We present a digital signature scheme which is based on the existence of any trapdoor permutation. Our scheme is secure in the strongest possible natural sense: namely, it is secure against existential forgery under adaptive chosen message attack.

* Department of Computer Science & Engineering, Mail Code 0114, University of California at San Diego, 9500 Gilman Drive, La Jolla, CA 92093. E-mail: mihir@cs.ucsd.edu. Work done while author was at MIT, supported in part by NSF grant CCR-87-19689.

† MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139. Supported in part by NSF grant DCR-84-13577 and ARO grant DAALO3-86-K-0171.

1 Introduction

1.1 The Diffie-Hellman Model of Digital Signatures

Fifteen years ago, Diffie and Hellman [DH] put forward a beautiful model for digitally signing. Their model is based on — and in some sense coincided with — their newly introduced notion of a *trapdoor* permutation, an extension of the notion of a *one-way* permutation.

Roughly, a permutation f is said to be one-way if it is computationally easy to evaluate, but computationally hard to invert. A one-way permutation f is trapdoor if it has an associated secret string, S_f, given which f becomes easy to invert.

Diffie and Hellman proposed using trapdoor permutations to achieve digital signatures as follows. Each user A selects a trapdoor permutation f_A "together with" its associated secret S_{f_A}. User A then publishes f_A and keeps secret S_{f_A}. Thus A is the only one who can efficiently invert f_A. To digitally sign a message (number) m, A computes the string $\sigma = f^{-1}(m)$. Given m and σ, any one can efficiently verify that σ is A's *digital signature* of message m by "looking up" A's published permutation f_A, computing $f_A(\sigma)$, and verifying that the result is indeed m.

At the time of their proposal, no one knew how to go about proving that trapdoor permutations exist (such a proof automatically entails that $P \neq NP$, something still out of reach), nor how to suggest concrete reasonable "candidate" trapdoor permutations. Soon afterwards, Rivest, Shamir, and Adleman [RSA] proposed an algebraic candidate, the RSA function, for which no efficiently inverting algorithm has yet been found (and may not exist).

1.2 Critique of the Model

Although quite elegant, the Diffie-Hellman model has some inherent limitations, brought to light by Goldwasser, Micali, and Yao [GMY]. For instance, they point out that it is impossible that a trapdoor permutation be hard to invert on all of its range. (For example the RSA function has the form $x^e \; mod \; n$, thus its inverse at 1 is 1.) At best one can prove that, with high probability, every efficient inverting algorithm fails to invert a trapdoor permutation at a point randomly selected in its range, but a small subset of the range will always exists for which inverting the permutation is easy. Security vanishes if the message set is contained in (or overlaps significantly with) the "easy" subset of the trapdoor permutation's range. Thus, even if we have proved that a given permutation is trapdoor, will we ever be able to prove that ASCII English avoids its range's easy subset?

Also, nowhere in the definition is it said that a trapdoor permutation f cannot enjoy additional properties. For instance, f may be "multiplicative" (like the RSA) in the sense that, given the value of f^{-1} at points x and y, computing $f^{-1}(xy)$ is easy. Mutatis mutandis, this means that, given "legitimately" obtained signatures of strings x and y, one may easily forge the signature of string xy. One may object that in the case of English messages, it is unlikely that the product of two messages is a proper English sentence. However, besides the fact that this may be hard to prove, in many applications we may need to be able to sign arbitrary numbers. What, then, would happen to the "security" of the scheme?

Finally, one could always generate legitimate signatures by choosing a strings σ and computing $m = f(\sigma)$.

In essence, [GMY] pointed out that the notion of security for digital signature schemes was far from being understood, and that the existence of trapdoor permutations (without adding extra assumptions — such as that multiplicativity does not hold, nor additivity, nor...) may not be sufficient to guarantee the existence of "secure" digital signature schemes.

1.3 The Notion of Security

What then is a satisfactory definition of security for digital signatures schemes? Since the above critique is not confined to any specific implementation but is rather about the Diffie and Hellman model itself, implicit in this question is the question of what the model should be.

A quite general definition of security for signature schemes was given by [GMY], as well as the first example of a scheme outside the Diffie and Hellman model. The "right" definition was found a year later by Goldwasser, Micali, and Rivest [GMR].

Informally, the [GMR] definition says that even an adversary who is granted a special experimentation session with the signer, in which the adversary asks and receives signatures of messages of his choice, cannot later efficiently forge the signature of any new message. This definition is formally presented in Section 3. For further motivation, and more history of these ideas, we address the reader to their original paper.

1.4 The Computational Assumptions Needed to Implement It

Having established the desired notion of security, it became very important to establish the computational assumptions necessary to implement it.

The existence of secure digital signatures was first proved assuming the computational difficulty of some outstanding mathematical problems; integer factorization in the case of [GMR] (actually [GMR] based their signature scheme on a general primitive which they called *claw-free pairs* and then showed how these could be implemented based on factoring).

Proving the security of a scheme implementing a basic primitive based on a purely mathematical assumption is not easy. Ever since the work of [GM] on public-key encryption, such proofs of security have had the form of a reduction. Namely, it is shown that if any algorithm exists for efficiently defeating the cryptographic scheme in question, then a (different) algorithm would exist for efficiently solving the underlying, mathematical problem. The difficulty of these proofs arises from the fact that they "reduce apples to oranges." For instance, an algorithm for attacking a digital signature scheme does not look related at all with any factoring algorithm: it has different goals, different inputs, etcetera.

This difficulty explains why the fundamental cryptographic primitives were first implemented based on the computational difficulty of some *specific* mathematical problem; that is, one possessing some additional property besides —say— trapdoorness. This may simplify the work of the scheme designer.

An attacker, of course, is always allowed to use any extra structure that might be present, although not explicitly assumed. If extra structure is explicitly assumed, it is at least available to the scheme designer as well.

Indeed, factoring (properly modified) yields a specific trapdoor permutation [W],[BBS], with some rich algebraic properties.

However, once we must resort to making assumptions, we better make the "smallest" ones. Trapdoor permutations may exist, but trapdoor permutations enjoying specific algebraic properties may not. In order to establish the existence of the basic cryptographic primitives, it is preferable (disregarding efficiency considerations) to assume an "abstract" trapdoor permutation, rather than on one possessing additional properties. Better yet is finding the minimal computational assumption needed. That is, finding conditions that are both necessary and sufficient to securely implementing our primitives.

1.5 Our Result

In this paper, we show that abstract trapdoor permutations are sufficient for digital signature schemes. Thus we show that the same complexity assumption hypothesized by Diffie and Hellman can be used (though by a totally different way) to achieve "perfect" security.

Let us say that we would have not devised our scheme without the work of [GMR] (from which we borrow definitions, notations, and several ideas) and the older work of Lamport [La].

1.6 Recent Improvements

Our basic digital signature scheme, together with other beautiful ideas of Merkle [M], have been successfully used by Naor and Yung [NY] and Rompel [R]. Naor and Yung show that even one-way permutations are sufficient for secure digital signatures. Rompel shows that one-way *functions* are actually sufficient for secure digital signatures. Since an easy argument of ours, reported in [R], shows that this condition is also necessary, the existence of a one-way function is the minimal complexity assumption for the existence of secure digital signatures schemes.

Since the works of [NY] and [R] heavily rely on ours, the present paper is also a quite effective introduction to their more complex schemes.

2 Notation and Conventions

2.1 Strings and Sequences

The empty string is denoted λ.

The length of a binary string s is denoted by $|s|$ while its i-th bit is denoted $(s)_i$. We use $(s)_{1\ldots i}$ to denote the string consisting of the first i bits of s; this is λ if $i = 0$.

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 $j \le i$ then $(a_1, \ldots, a_j)$ is called an *initial segment* of a.

2.2 Notation for Probabilistic Algorithms

We use [GMR]'s notation and conventions for probabilistic algorithms.

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 we write "$A(\cdot,\cdot)$", and so on. If A is a probabilistic algorithm then, for any input i the notation $A(i)$ refers to the probability space which to the string σ assigns the probability that A, on input i, outputs σ (in the special case that A takes no inputs, A refers to the algorithm itself whereas the notation $A()$ refers to the probability space obtained by running A on no inputs).

If S is a probability space we denote its support (the set of elements of positive probability) by $[S]$.

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 $f(g(x, y, \cdots))$.

If S is a probability space 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 $\mathbf{P}_S[e]$). In the case that $[S]$ consists of only one element e we may write $x \leftarrow e$.

For probability spaces $S, T, \ldots$, the notation

$$\mathbf{P}\left[\, p(x, y, \cdots) \,:\, x \leftarrow S; y \leftarrow T; \cdots \right]$$

denotes the probability that the predicate $p(x, y, \cdots)$ is true after the (ordered) execution of the algorithms $x \leftarrow S$, $y \leftarrow T$, etc.

If S is a finite set we will identify it with the probability space which assigns to each element of S the (uniform) probability $\frac{1}{|S|}$. Thus $x \leftarrow S$ denotes the operation of selecting an element of S uniformly at random (again in the case that the set is of only one element e we may write $x \leftarrow e$ rather than $x \leftarrow \{e\}$).

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

3 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 the notion of security against adaptive chosen message attack [GMR].

3.1 Components of a Signature Scheme

A digital signature scheme has the following components:

- A *security parameter* k. This is chosen by the user when he creates his public and secret keys, and determines overall security, the length and number of messages, and the running time of the signing algorithm.
- A *message space*. This 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 *key generation algorithm* $\mathcal{KG}$. This is a probabilistic polynomial time algorithm which can be run by any user to produce, on input 1^k, a pair (PK, SK) of matching public and secret keys.
- A *signing algorithm* $\mathcal{S}$. This is a probabilistic polynomial time algorithm which given a message m and a pair (PK, SK) of matching public and secret keys outputs 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 *verification algorithm* $\mathcal{V}$. This is a polynomial time algorithm which given S, m, and PK outputs `true` if S is a valid signature for the message m with respect to the public key PK, and `false` otherwise.

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.

3.2 Security of a Signature Scheme

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*.

In an adaptive chosen message attack a forger uses the signer A to obtain signatures of messages of his choice. He is allowed to choose these messages not only as a function of A's public key but as a function of the signatures returned by A in response to the forger's previous requests. That is, the forger begins by picking some message m_1 as a function of A's public key PK and obtaining from A a signature S_1 of it. As a function of PK, m_1 and S_1 he now chooses m_2 and gets a signature S_2 of it. This goes on for a polynomial (in the security parameter) number of messages.

From the knowledge so gathered the forger 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.

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

(1) engages in a conversation with the legal signer $\mathcal{S}$, requesting and receiving signatures with respect to PK for messages of his choice (the adaptive chosen message attack),
(2) then outputs a pair (m, S) (an attempt at existential forgery).

We say $\mathcal{F}$ is *successful* if

(1) S is a signature of m with respect to PK (i.e. $\mathcal{V}(S, m, PK) = \texttt{true}$), and
(2) a signature of m was not requested of $\mathcal{S}$ in the adaptive chosen message attack.

Definition 3.1 Let Q be a polynomial. We say that a signature scheme is *Q-forgeable* 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 $\mathcal{KG}$) and over the coin tosses of $\mathcal{F}$ and $\mathcal{S}$.)

Definition 3.2 A signature scheme is *secure* if for all polynomials Q it is the case that the scheme is not Q-forgeable.

4 Trapdoor Permutations

Informally, a family of of trapdoor permutations is a family of permutations such that

- it is easy, given an integer k, to randomly select a permutation f which has security parameter k, together with some "trapdoor" information associated with f
- f is easy to compute, and, given the trapdoor information, so is f^{-1}; but without the trapdoor information, f is "hard" to invert.

But what exactly does it mean for f to be hard to invert? No finite function is hard to invert, so the formalization must be in terms of families of functions. Moreover no function is hard to invert at *all* points in its range, since one could always have an algorithm which when asked to invert f at y, evaluates f at a few fixed points and checks whether any of these evaluations yields y. We ask rather that f be hard to invert at a random input.

4.1 A Simple Definition

Definition 4.1 A triplet (G, E, I) of probabilistic polynomial time algorithms (the *generating*, *evaluating* and *inverting* algorithms respectively) 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 $A(\cdot,\cdot,\cdot)$, for all c and sufficiently large k,

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

In informal discussions we will omit explicit mention of the generator and talk of f being a "trapdoor permutation". It is to be understood that $f(\cdot) = E(x,\cdot)$ for some $(x,y) \in [G(1^k)]$ where (G,E,I) is the underlying trapdoor permutation generator.

Although simpler, our definition is potentially less general than that of [GMR]. The difference lies in the nature of the domain on which a trapdoor permutation is defined. We ask that the domain be $\{0,1\}^k$ when the security parameter is k, while [GMR] only require that it be a set which can be *sampled*; their generator produces with each permutation an algorithm which can produce a random point of the domain. We utilize this difference in our scheme.

However our definition is without loss of generality in the sense that it does capture all known candidates for trapdoor permutations. A simple, general construction due to Yao [Y] fits RSA and other candidates into our scenario. An informal description of this construction follows.

4.2 Using Yao's Construction

Take for example the RSA function [RSA], the most popular candidate for a trapdoor permutation. The domain of the trapdoor permutation f when the security parameter is k is $D = Z_k^*$, a subset of the k bit strings. The function f is assumed, say, hard to invert on a $1 - \frac{1}{k^c}$ fraction of D.

First extend f to $\{0,1\}^k$ by defining

$$\bar{f}(x) = \begin{cases} f(x) & \text{if } x \in D \\ x & \text{otherwise.} \end{cases}$$

This new function is a permutation on $\{0,1\}^k$ but might be easy to invert on a polynomial fraction of the domain. Yao's cross product construction can now be used to pump up the security. We define the function F on

$$\underbrace{\{0,1\}^k \times \ldots \times \{0,1\}^k}_{n}$$

by

$$F(x_1, \ldots, x_n) = (\bar{f}(x_1), \ldots, \bar{f}(x_n)) \; ;$$

F is still a permutation, and Yao shows that by choosing n to be an appropriate polynomial in k, it can be made to satisfy (2) of definition 4.1.

The same construction works for the trapdoor permutations of [BBS]. In general, the construction can be applied whenever

- the domain of f is a subset of $\{0,1\}^k$ of size at least a polynomial fraction of $\{0,1\}^k$
- there is a (polynomial time) algorithm to determine whether a given point lies in the domain.

5 An Overview of the Scheme

We present here an overview of our scheme and a sketch of the proof of security. For simplicity we will for the moment completely disregard efficiency.

5.1 Background

Lamport [La] suggested the following method for signing a single bit. The signer makes public a trapdoor permutation f and a pair of points α^0 and α^1, and keeps secret f^{-1}. The signature of a bit $b \in \{0,1\}$ is then $f^{-1}(\alpha^b)$.

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 a different way of signing a single bit.

In the scheme described below, and then in more detail in Section 6, 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 which directly uses the Lamport format but signs new *functions* instead is described briefly in Section 8. (The latter version of the scheme was also presented in [BeMi]).

5.2 The Signature Scheme

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

$$PK = (f_{0,0}, f_{0,1}, \ldots, f_{k,0}, f_{k,1}, \alpha_0)$$

where the $f_{i,j}$ are trapdoor permutations with security parameter k and α_0 is a random k bit string (we refer to k bit strings equivalently as *points* or *seeds*). His secret key is the trapdoor information $f_{i,j}^{-1}$ $(i, j = 0, \ldots, k)$.

Suppose the message to be signed consists of a single bit b. The signer executes the following two steps:

(1) *Sign b:* He reveals $f_{0,b}^{-1}(\alpha_0)$

(2) *Regenerate the Public Key:* He picks at random a new seed α_1 and sends it to the receiver. He signs this seed by revealing for each $i = 1, \ldots, k$, either $f_{i,0}^{-1}(\alpha_0)$ or $f_{i,1}^{-1}(\alpha_0)$ 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 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_0 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 $\alpha, \alpha_1, \alpha_2, \ldots$ in which each element of the chain is used to sign its successor.

We note that a seed is never reused: α_{i-1} is used to sign one message bit and α_i and then never touched again.

5.3 Why is this Secure?

Suppose $\mathcal{F}$ is a successful forger (as described in Section 3.2). 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 at z.

A creates a public key $PK = (f_{0,0}, f_{0,1}, \ldots, f_{k,0}, f_{k,1}, \alpha_0)$ in which $f_{n,c} = g$ for a randomly chosen $n \in \{0, \ldots, k\}$ and $c \in \{0,1\}$ and all the other functions are obtained by running the generator (so A knows their inverses). In the course of signing A will generate a list of random seeds of the form $\alpha_l = g(\beta_l)$, except for some one (random) 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 $f_{n,c}^{-1}(z)$.

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 since $\mathcal{F}$ does not know where z has been placed it will happen with probability at most $1/2$. So A succeeds in responding to all $\mathcal{F}$'s requests with probability $\geq 1/2$.

By assumption $\mathcal{F}$ will now return the signature of a message not signed previously 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.

6 The Signature Scheme

Let (G, E, I) be a trapdoor permutation generator . With some abuse of language we will often call x a function and identify it with $E(x, \cdot)$.

6.1 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 $\vec{x} = (x_0^0, x_0^1, \ldots, x_k^0, x_k^1)$. Let $\alpha, \alpha' \in \{0,1\}^k$.

Definition 6.1 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 $i = 1, \ldots, k$.

Intuitively, α' (which we think of as already having been authenticated) is authenticating α (with respect to $\vec{x}$) via the Lamport format. For each bit $(\alpha)_i$ of α the seed authenticator $\langle \alpha'; \alpha \rangle_{\vec{x}}$ includes a value z_i. This value is $f_{i,0}^{-1}(\alpha')$ if $(\alpha)_i = 0$ and $f_{i,1}^{-1}(\alpha')$ if $(\alpha)_i = 1$ (where $f_{i,j}(\cdot) = E(x_{i,j}, \cdot)$).

Bits are authenticated similarly:

Definition 6.2 A *bit authenticator* $\langle \alpha'; b \rangle_{\vec{x}}$ is a tuple of strings (α', b, z) such that $b \in \{0,1\}$ and $E(x_0^b, z) = \alpha'$.

Notice that we have $k+1$ pairs of functions $x_{i,j}$ so that a single α' can authenticate up to $k+1$ bits. We use the first (0-th) pair of functions to authenticate a bit and the rest to authenticate a k-bit seed.

Definition 6.3 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 the 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 $y_0^0, y_0^1, \ldots, y_k^0, y_k^1$.

Definition 6.4 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 6.5 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.

6.2 Generating Keys

The key generation algorithm $\mathcal{KG}$ 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 $PK = (1^k, \vec{x}, \alpha)$ where $\vec{x} = (x_0^0, x_0^1, \ldots, x_k^0, x_k^1)$.
(4) Output the secret key $SK = \vec{y} = (y_0^0, y_0^1, \ldots, y_k^0, y_k^1)$.

6.3 Signatures

Definition 6.6 A *signature* of a message $m \in \mathcal{M}_k$ with respect to a public key $PK = (1^k, \vec{x}, \alpha)$ 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 1 (right) shows a schema of a signature for a message m with respect to a public key $(1^k, \vec{x}, \alpha_0)$; here $F^i = \langle \alpha_{i-1}; \alpha_i \rangle_{\vec{x}}$ $(i = 1, \ldots, pk)$ and $B^i = \langle \alpha_{(p-1)k+i}; (m)_i \rangle_{\vec{x}}$ $(i = 1, \ldots, k)$.

6.4 The Signing Algorithm

Let $PK = (1^k, \vec{x}, \alpha_0)$ 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 $m_i \in \mathcal{M}_k$, the signing procedure $\mathcal{S}$ does the following:

(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 $(F^{l+1}, \ldots, F^{l+k})$.

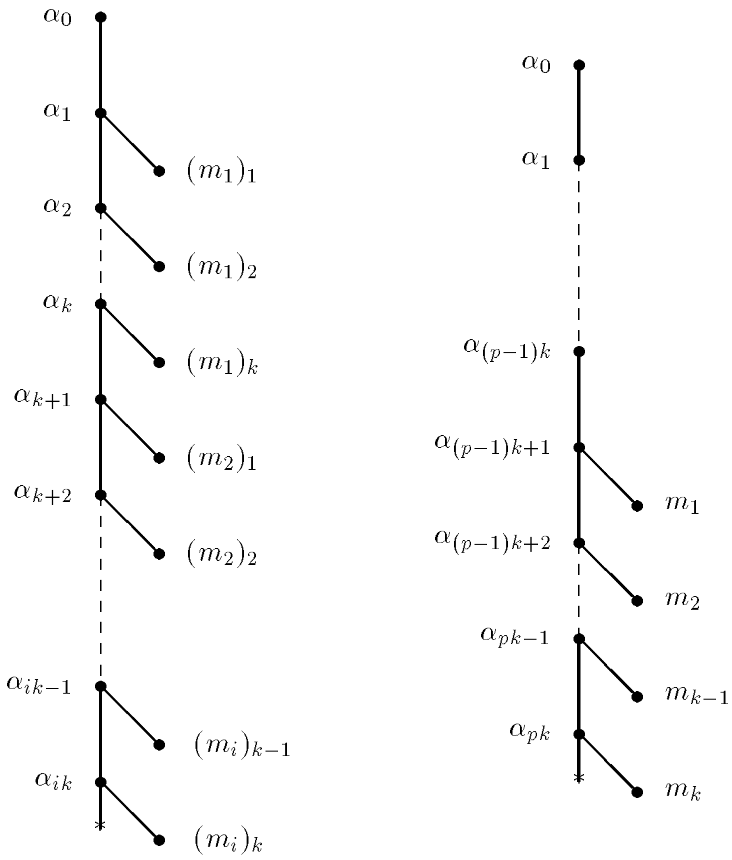

Figure 1: A signature corpus (left), and a signature of a message m (right)

(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 (left) shows a schema of the data structure constructed by the signing procedure as described above. This structure will be called a signature corpus in Section 7.

6.5 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 6.6 only requires knowledge of the public key.

7 Proof of Security

7.1 The Signature Corpus

Definition 7.1 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 7.2 Let $Z = (F, B, M)$ be either a single signature or a signature corpus, relative to a public key $PK = (1^k, \vec{x}, \alpha_0)$, 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 $M(Z) = m$).

7.2 Extracting Information From a Forgery

As indicated in the overview of Section 5.3, forgery must eventually be used to extract information about the inversion of a trapdoor function. The preliminary definitions and lemmas here are devoted to characterizing the structure of a forgery relative to a given corpus.

Lemma 7.3 *Let C be a signature corpus relative to a public key $\mathrm{PK} = (1^k, \vec{x}, \alpha)$ 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 $PK = (1^k, \vec{x}, \alpha)$ 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 7.4 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 7.5 *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 7.3. The proof breaks down into the cases provided by Lemma 7.3, 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 7.3 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 7.4 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 7.4 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 7.4 says that (α', x_i^j) is unused in C. ∎

7.3 Proof of Security

We are now ready to prove

Theorem 7.6 *Under the assumption that* (G, E, I) *is a trapdoor permutation generator the above signature scheme is secure.*

(see Definition 3.2 for the definition of security).

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 successful in forging with probability $\geq \frac{1}{Q(k)}$ on input a public key chosen according to the distribution induced by $\mathcal{KG}$. 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 $E(x, \cdot)^{-1}(z)$.

Since $\mathcal{F}$ is probabilistic polynomial time there is a polynomial S_F such that the number of signatures requested by $\mathcal{F}$ is at most $S_F(k)$. We now define A to operates as follows on input $1^k, x, z$:

(1) Let $n \leftarrow \{0, \ldots, k\}$, $c \leftarrow \{0, 1\}$, and $t \leftarrow \{0, \ldots, kS_F(k)\}$.
(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 $kS_F(k)$ random k bit strings $\beta_0, \ldots, \beta_{t-1}, \beta_{t+1}, \ldots, \beta_{kS_F(k)}$, 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 $(\alpha_0, \alpha_1, \ldots, \alpha_{kS_F(k)})$.

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

(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 successfully 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 7.5 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 7.7 *The public key* PK *created in step 4 has the same distribution as that induced on public keys by the key generation algorithm $\mathcal{KG}$ of Section 6.2.*

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 $\mathcal{KG}$. ■

Lemma 7.8

(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 successfully 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 $\mathcal{F}$ up to this crucial point have the same distribution as the legal signer would have generated. 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 7.9 *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 7.4. Otherwise, since the signing was accomplished, either (1) or (3) must hold. ■

By Lemma 7.5, $u = E(x_i^j,\cdot)^{-1}(\alpha')$ for some pair (α', x_i^j) unused in C. We would like the pair to actually be (z,x), for then $u = E(x,\cdot)^{-1}(z)$. We consider the probability that $u = E(x,\cdot)^{-1}(z)$ conditioned on the event that z is in $P(C)$ and (z,x) is unused in C. By the randomization of the n and t parameters (step 1) this probability is

$$\geq \frac{1}{(1+k)(1+kS_F(k))} \, .$$

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

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

contradicting the fact that G is a trapdoor permutation generator. This completes the proof of Theorem 7.6.

8 The Dual Scheme

An interesting feature of our scheme is a "duality" between the roles of functions and seeds. The roles as described in the scheme of the previous sections can be interchanged to yield an equivalent scheme which keeps a fixed number of seeds in the public key and signs new *functions* as needed.

The duality is real enough to make the structure and description of the schemes, as well as the proof of security, entirely symmetric, and the ability to sign new functions rather than new seeds is unusual enough to merit a little description. In fact, it is this dual scheme that was presented in detail in [BeMi], and the interested reader can obtain details from there.

We point out, though, that the scheme of Section 6 is the far more natural one for implementation. In practice it is of course easier to generate a random element of $\{0,1\}^k$ (which just consists of k coin flips) than it is to run a possibly quite complex generator to get a trapdoor function (a typical generator is attempting to find certified primes of some length and so forth, a comparatively expensive operation). The dual scheme is thus likely to be a good deal less efficient.

8.1 Description

We outline the structure of the dual scheme. The public and secret keys are of the form

$$\begin{aligned} PK &= (1^k, x_0, \vec{\alpha}) \\ SK &= y_0 \end{aligned}$$

where $\alpha = (\alpha_0^0, \alpha_0^1, \ldots, \alpha_k^0, \alpha_k^1)$ is a vector of $2k+2$ randomly chosen k bit strings, and $(x_0, y_0) \in [G(1^k)]$.

Each signing step consists of signing a bit of the message and a new k bit trapdoor permutation x_l to take the place of the trapdoor permutation x_{l-1} of the previous stage (x_0 is used in the first stage). The $k+1$ bits consisting of the message bit and the k bits of x_l are signed by sending either $I(y_l, \alpha_i^0)$ or $I(y_l, \alpha_i^1)$ depending on whether the i-th bit was a 0 or a 1. Note that the signer gets new trapdoor permutations by running the generator; he thus knows and preserves the inverses while signing and revealing the functions themselves. A total of $2k+2$ seeds stay fixed in the public key, while random trapdoor permutations are generated and signed as needed to propagate the signatures; the roles of points and functions relative to the original scheme (Section 6) are effectively interchanged.

To illustrate in a little more detail, we would have what we could call *function authenticators*

$$\langle x'; x \rangle_{\vec{\alpha}} = (x', x, z_1, \ldots, z_k)$$

such that $E(x', z_i) = \alpha_i^{(x)_i}$, where $x, x' \in [G(1^k)]$; bit authenticators would now be of the form

$$\langle x'; b \rangle_{\vec{\alpha}} = (x', b, z)$$

with $E(x', z) = \alpha_0^b$. Spines would now consist of sequences of function authenticators, and so forth. The entire scheme of Section 6 would carry over with essentially just a change of terminology.

8.2 Proof of Security for the Dual

It is easy to see that the proof of Theorem 7.6 for the dual is just symmetric. To illustrate a little: A would create the public key by placing the given point z in $\vec{\alpha}$, and making the rest of the points of α of the form $E(x, \beta_i^j)$. A would sign new functions obtained by running G except at some one random stage when it would use x. It could then simulate the signing and get $E(x, \cdot)^{-1}(z)$ from the forger in the same manner as before.

9 Using Tree Structures

A key tool in improving the efficiency of our scheme and in eventually getting a memoryless version is the use of tree structures in the style of [GMR]. We describe in this section the nature of the basic tree based scheme.

Henceforth the scheme of Section 6 will be referred to as the linear scheme.

9.1 Structures for the Tree Scheme

In the tree scheme, the public and secret keys are of the form

$$\begin{aligned} PK &= (1^k, \vec{x}, \vec{x}_0, \vec{x}_1, \alpha_\lambda, \beta) \\ SK &= (\vec{y}, \vec{y}_0, \vec{y}_1), \end{aligned}$$

where

$$\begin{aligned} \vec{x} &= (x^0, x^1) \\ \vec{x}_0 &= (x_{0,1}^0, x_{0,1}^1, x_{0,2}^0, x_{0,2}^1, \ldots, x_{0,k}^0, x_{0,k}^1) \\ \vec{x}_1 &= (x_{1,1}^0, x_{1,1}^1, x_{1,2}^0, x_{1,2}^1, \ldots, x_{1,k}^0, x_{1,k}^1) \end{aligned}$$

are trapdoor permutations, $\vec{y}$, $\vec{y}_0$, $\vec{y}_1$ are their respective inverses, α_λ is a k bit seed, and the parameter β defines a *signature bound*: $S_B = 2^\beta$ is a bound on the total number of signatures that can be signed with respect to the public key.

While signing, a binary tree of seeds is created, with a parent authenticating two children. The new children seeds are signed in the same manner as the previous scheme, using $\vec{x}_0$ as the source to sign the left child, and $\vec{x}_1$ as the source for the right child.

Seed authenticators thus come in two varieties, a left and a right:

$$\begin{aligned}\langle \alpha'; \alpha \rangle_{\vec{x}_0} &= (\alpha', \alpha, 0, z_1, \ldots, z_k)\\ \langle \alpha'; \alpha \rangle_{\vec{x}_1} &= (\alpha', \alpha, 1, z_1, \ldots, z_k)\end{aligned}$$

where $\alpha, \alpha' \in \{0,1\}^k$ and $E(x_{j,i}^{(\alpha)_i}, z_i) = \alpha'$ for all $i = 1, \ldots, k$. Bit authenticators are with respect to $\vec{x}$:

$$\langle \alpha'; b \rangle_{\vec{x}} = (\alpha', \alpha, z)$$

with $E(x^b, z) = \alpha'$. The terminology of roots, children, and sources of authenticators (Definition 6.3), as well as that of spines and attaching (Definitions 6.4, 6.5) remains the same. We can then define a signature,

Definition 9.1 A *signature* of a message $m \in \mathcal{M}_k$ with respect to a public key $PK = (1^k, \vec{x}, \vec{x}_0, \vec{x}_1, \alpha_\lambda, \beta)$ is a triple (F, B, m) where $F = (F^1, \ldots, F^\beta, F^{\beta+1}, \ldots, F^{\beta+k})$ is a spine and $B = (B^1, \ldots, B^k)$ is a sequence of bit authenticators such that

- B is $(\beta+1)$-attached to F.
- F starts at α_λ.
- For all $i = 1, \ldots, k$ the child of B^i is $(m)_i$ and the source of B^i is $\vec{x}$.
- For all $i = 1, \ldots, \beta$ the source of F^i is either $\vec{x}_0$ or $\vec{x}_1$.
- For all $i = \beta+1, \ldots, \beta+k$ the source of F^i is $\vec{x}_0$.

9.2 Signing

We now describe the signing procedure. Let

$$\begin{aligned}PK &= (1^k, \vec{x}, \vec{x}_0, \vec{x}_1, \alpha_\lambda, \beta)\\ SK &= (\vec{y}, \vec{y}_0, \vec{y}_1),\end{aligned}$$

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 $m_0, \ldots, m_{i-1}$ and kept track of the signatures $S_0 = (F_0, B_0, m_0), \ldots, S_{i-1} = (F_{i-1}, B_{i-1}, m_{i-1})$ of these messages.

The integer i $(0 \le i < 2^\beta)$ will be represented here as a binary string of exactly β bits; that is, the representation of i is i in binary padded with leading zeroes to bring the total length to exactly β. With the notation of Section 2.1, $(i)_{1\ldots t}$ then denotes the first t bits of this string.

To compute a signature $S_i = (F_i, B_i, m_i)$ for m_i, where $i < 2^\beta$ and $m_i \in \mathcal{M}_k$, $\mathcal{S}$ now performs the following steps:

(1) If $i = 0$ then let $s = \lambda$. Otherwise, let s be the longest common prefix of i and $i-1$.

(2) Select at random $\beta - |s|$ seeds $\alpha_{(i)_{1\ldots|s|+1}}, \ldots, \alpha_{(i)_{1\ldots\beta}}$ from $\{0,1\}^k$, and form the seed authenticators

$$F^{(i)_t} = \langle \alpha_{(i)_{1\ldots t-1}}; \alpha_{(i)_{1\ldots t}} \rangle_{\vec{x}_{(i)_t}}$$

for $t = |s|+1, \ldots, \beta$.

(3) Let F be the spine $(F^{(i)_1}, F^{(i)_{1\ldots 2}}, \ldots, F^{(i)_{1\ldots\beta}})$ (thus F consists of the first $|s|$ seed authenticators from F_{i-1} (if $i > 0$) followed by the $\beta - |s|$ authenticators created in step 2).

(4) Now form a signature of m_i in the style of the linear scheme. That is, select k random seeds $\gamma_1, \ldots, \gamma_k$. Let $\gamma_0 = \alpha_{(i)_{1\ldots\beta}}$, and let

$$\begin{aligned}H^j &= \langle \gamma_{j-1}; \gamma_j \rangle_{\vec{x}_0}\\ B^j &= \langle \gamma_j; (m_i)_j \rangle_{\vec{x}}\end{aligned}$$

for $j = 1, \ldots, k$.

(5) Let $F_i = F * (H^1, \ldots, H^k)$, $B_i = (B^1, \ldots, B^k)$, and output $S_i = (F_i, B_i, m_i)$ as the signature of m_i.

In the course of signing, $\mathcal{S}$ is thus building a binary tree of height β whose nodes are labeled by seeds. The root is labeled with α_λ, and the left and right children of a node with label α_s are labeled α_{s0} and α_{s1} respectively. Seed authenticators link a parent to its children:

$$\begin{aligned} F^{s0} &= \langle \alpha_s ; \alpha_{s0} \rangle_{\vec{x}_0} \\ F^{s1} &= \langle \alpha_s ; \alpha_{s1} \rangle_{\vec{x}_1} \quad . \end{aligned}$$

The signature of the i-th message consists of the seed authenticators which form the path to the i-th leaf of this tree (numbering the leaves left to right, beginning at 0^β), together with a further linear chain in the style of the previous scheme.

9.3 Security

The argument to prove

Theorem 9.2 *Under the assumption that (G, E, I) is a trapdoor permutation generator the above tree based signature scheme is secure.*

is based on ideas entirely similar to those used in the proof of Theorem 7.6, and is complicated more by cumbersome notation than anything else. We thus omit it here.

9.4 Advantages of the Tree Scheme

The tree scheme not only produces more compact signatures, but has the advantage that the size of a signature is independent of the sizes of previous signatures. In order to discuss signature sizes more precisely, it is convenient to talk in terms of the *length* of a signature, a quantity easily visualized.

Definition 9.3 The *length* of a signature $S = (F, B, m)$ (in either the tree scheme or the linear scheme), denoted *length*(S), is the number of seed authenticators in F.

The size of a signature S (in bits) is then $O(k^2 \cdot \mathit{length}(S))$, in either scheme.

For the tree scheme with signature bound S_B, the signature of a message m is always of length $|m| + \log S_B(k)$ which is $k + \log S_B(k)$ for $m \in \mathcal{M}_k$. In the linear scheme, the signature of the i-th message m_i reaches a length of $\sum_{j=1}^{i} |m_j| = ki$ and thus if S_B messages were to be signed the signature lengths reach $kS_B(k)$.

10 Variations and Improvements

10.1 Arbitrary Length Messages

The assumption that messages are always of length equal to the security parameter was made to simplify the proof, and can easily be removed. Messages of any length bounded by a fixed polynomial in k are allowed, as long as they come from a *subsequence free set*;

Definition 10.1 A set S of binary strings is *subsequence free* if any sequence $s_1, \ldots, s_i$ of strings from S has the property that $s \in S$ is a substring of $s_1 \ldots s_i$ if and only if $s = s_j$ for some j.

Note that if a set S is subsequence free then it is automatically prefix free, but not vice-versa.

It is easy to encode arbitrary non-empty strings so that the resulting set is subsequence free. For example, encode a string by replacing each 0 by 00, each 1 by 11, and finally adding 01 to the beginning and end. The new string is about twice as long as the old. This is much better than encoding into strings of length k (at least for $k \geq 3$).

10.2 A Memoryless Version

[GMR] describe a method to make their scheme memoryless; they attribute the basic idea to Levin and improvements to [Go]. Another memoryless version of the scheme is due to [Gu]. We note here that the former set of techniques can be applied to make our scheme memoryless as well. The basic tool is the use of pseudo-random functions [GGM], whose existence is implied by our assumptions. A function from a pseudo-random collection is put in the secret key and used to *compute* the seeds in the tree ([GMR] call them *roots*) in a specific manner. Further, a random branch of the tree is chosen to sign the next message rather than using the branches in order from left to right.

References

[BeMi] Bellare, M., and S. Micali, "How to Sign Given Any Trapdoor Function," Proceedings *of the* 20th *Annual Symposium on Theory of Computing*, ACM, 1988.

[BBS] Blum, L., M. Blum, and M. Shub, "A Simple Unpredictable Pseudo-Random Number Generator," *SIAM Journal on Computing* **15**(2), 364-383, May 1986.

[BlMi] Blum, M., and S. Micali, "How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits," *SIAM Journal on Computing* **13**(4), 850-864, November 1984.

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

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

[GGM] Goldreich, O., S. Goldwasser, and S. Micali, "How To Construct Random Functions," *Journal of the ACM* **33**(4), 792-807, October 1986.

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

[GMR] Goldwasser, S., and S. Micali, and R. Rivest, "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks," *SIAM Journal on Computing* **17**(2), 281-308, April 1988.

[GMY] Goldwasser, S., S. Micali, and A. Yao, "Strong Signature Schemes," Proceedings *of the* 15th *Annual Symposium on Theory of Computing*, ACM, 1983.

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

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

[M] Merkle, R., "A Digital Signature Based on a Conventional Encryption Function," *Advances in Cryptology – Crypto 87 Proceedings*, Lecture Notes in Computer Science Vol. 293, C. Pomerance ed., Springer-Verlag, 1987.

[NY] Naor, M., and M. Yung, "Universal One-Way Hash Functions and their Cryptographic Applications," Proceedings *of the* 21st *Annual Symposium on Theory of Computing*, ACM, 1989.

[RSA] Rivest, R., A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM* **21**(2), 120-26, February 1978.

[R] Rompel, J., "One-Way Functions are Necessary and Sufficient for Secure Signatures," Proceedings *of the* 22nd *Annual Symposium on Theory of Computing*, ACM, 1990.

[W] Williams, H.C., "A Modification of the RSA Public-Key Cryptosystem," *IEEE Trans. Inform. Theory, IT-26*, 726-729, 1980.

[Y] Yao, A. C., "Theory and Applications of Trapdoor Functions," Proceedings *of the* 23rd *Symposium on Foundations of Computer Science*, IEEE, 1982.