Universal One-Way Hash Functions and their Cryptographic Applications [*]

Moni Naor[†]　　Moti Yung[‡]

Revised March 13, 1995

Abstract

We define a *Universal One-Way Hash Function* family, a new primitive which enables the compression of elements in the function domain. The main property of this primitive is that given an element x in the domain, it is computationally hard to find a different domain element which collides with x. We prove constructively that universal one-way hash functions exist if any 1-1 one-way functions exist.

Among the various applications of the primitive is a *One-Way based Secure Digital Signature* Scheme which is existentially secure against adoptive attacks. Previously, all provably secure signature schemes were based on the stronger mathematical assumption that *trapdoor* one-way functions exist.

Key words. cryptography, randomized algorithms

AMS subject classifications. 68M10, 68Q20, 68Q22, 68R05, 68R10

[*]Part of this work was done while the authors were at the IBM Almaden Research Center. The first author was supported in part by NSF grant CCR-88 13632. A preliminary version of this work appeared in Proc. of the 21st ACM Symposium on Theory of Computing

[†]Incumbent of the Morris and Rose Goldman Career Development Chair, Dept. of Applied Mathematics and Computer Science, Weizmann Institute of Science, Rehovot 76100, Israel. Most of this work performed while at the IBM Almaden Research Center. Research supported by an Alon Fellowship and a grant from the Israel Science Foundation administered by the Israeli Academy of Sciences. E-mail: naor@wisdom.weizmann.ac.il.

[‡]IBM Research Division, T.J Watson Research Center, Yorktown Heights, NY 10598, USA. E-mail: moti@watson.ibm.com.

1 Introduction

Consider an environment in which users share computer programs that reside in the common (read/write) space. We assume that a small read-only area is available in the shared memory. To prevent computer viruses from modifying the programs, the users would like to authenticate the programs before their use. A possible way to do it is to use the read-only area as a common security server which stores the hash value of the files of the common space, where the hash function h itself is publicly known and can be stored in the read-only space as well. The property required from the hash function h is that for a given value x it is computationally hard to find a y such that $h(y) = h(x)$ and $y \neq x$.

We call the hashing of a large file a *public fingerprint*. Notice that this scheme does not require the user to own a private secure space, the compression method is public, and (in the spirit of modern cryptography) private keys are not required. Furthermore, it can provide authentication even to a new or a casual user; this cannot be achieved by previously suggested (private) fingerprinting techniques [34, 29]. This scenario exemplifies the setting of this paper which combines cryptographic security and data compression; as we shall see, this setting includes a variety of applications.

We consider cryptographic primitives that implement such hash functions: In the first part of the paper we give a computational complexity definition of a new primitive, which we call *universal one-way hash functions*, and then show how to construct the primitive based on any 1-1 one-way function. In the second part we present applications; in particular, we show how to use the primitive to construct a digital signature which is provably secure and is based on the existence of any 1-1 one-way functions.

In the rest of the introduction, we discuss cryptographic assumptions and primitives in section 1.1, and define and provide the history of digital signatures in section 1.2.

1.1 Cryptographic Assumptions and Primitives

A current research program in cryptography is to provide constructions of basic primitives under assumptions that are as general as possible. Usually, these primitives are first introduced and implemented based on specific assumptions (e.g. factoring of a specific family of composite numbers is hard); these implementations rely on certain algebraic properties of the assumption involved. On the other hand, basing cryptography on general assumptions provides a unified cohesive complexity-theoretic formalism and it enriches the choice of candidates for underlying mathematical tools for implementations. Therefore, general assumptions are appealing to both theorists and practitioners.

Diffie and Hellman [6], who initiated public-key cryptography in 1976, suggested the general tools of one-way functions and one-way trapdoor functions. Basically, a function F is one-way if for a random x, given $F(x)$ it is hard to compute x. A function E is one-way trapdoor if it is one-way, and in addition, there exists a secret piece of information D, called a trapdoor, which represents the inverse function, such that $D(E(x)) = E(D(x)) = x$ and the knowledge of D enables easy inversion. The idea of Diffie and Hellman has since been formalized, and proof techniques based on complexity theory have been developed. In particular, Yao [35] has initiated the research program of basing cryptographic primitives on general assumptions.

Examples of cryptographic primitives (in addition to the new one mentioned above) are: (a) secure message sending [6, 31, 30, 13], (b) cryptographically secure pseudo-random generation [33, 2], and (c) general zero-knowledge interactive proofs [14]. In recent years, all these primitives have been constructed under formal general assumptions: message sending on trapdoor one-way functions [35, 17], pseudo random bit generator [35, 22, 11, 17], and general zero-knowledge interactive proofs [12, 20, 27] on one-way functions.

1.2 One-way hash

The need for a primitive that allows hashing so that it would be hard to find collisions was recognized since the earlier days of modern cryptography: Diffie and Hellman [6] mention such functions and Rabin [28] lists required properties of such functions. Other examples of work involving such functions are Merkle's [24, 25, 23] and Damgard [5]. One usage of such functions was in conjunction with digital signatures: instead of signing a long message, apply the hash function and sign the result.

The property that all previous researcher looked for was that given the description of the hash function h it should be hard to find $x \neq y$ such that $h(x) \neq h(y)$. We deviate from this scenario and relax the rules somewhat. In our game, first x is chosen arbitrarily by an adversary, then and h is chosen at random and the adversary is challenged to come up with a $y \neq x$ such that $h(x) = h(y)$. This relaxation proves to be very useful: it allows construction given any 1-1 one-way function, yet it is sufficiently strong to support applications such as public fingerprints and digital signatures. No provably secure general construction for one-way hash functions that previous researchers considered is known.

1.3 Digital Signatures

The history of digital signature started by Diffie and Hellman [6] where a signature scheme based on trapdoor one-way function was provided. Each user has a public key E and its secret inverse D as a private key. The signer sends the message M and signs by applying and sending $D(M)$, any receiver can verify the signature by computing $E(D(M))$ and checking that this value matches the message M. The above scheme was developed without a precise notion of security; the argument was that since inversion is hard to compute, a forgery is intractable and the system is secure. The first implementations of public-key cryptography (the Merkle-Hellman, RSA and Rabin's schemes [26, 31, 30]) gave signature schemes of this kind.

Following [6], signature systems design has become an extensive field of research (see [15]); we concentrate here only on provably secure systems. The first scheme to deal formally with the notion of security of signature scheme was suggested by Goldwasser, Micali and Yao [16] who also pointed out flaws in the Diffie-Hellman scheme. They based their probabilistic scheme on the problem of factoring. Then, the strongest known definition of security was formalized by Goldwasser, Micali, and Rivest [15]; they defined what it means for a system to be *existentially unforgeable under an adaptive chosen plaintext attack*; (this is what we call "secure" in the rest of the paper). This is an attack by an adversary (forger) who initially computes a plaintext and receives from the signature algorithm a corresponding valid signature; this is repeated in an adaptive fashion, for polynomially many iterations. Then the forger has to produce, without the cooperation of the signature algorithm, an extra signature for a message that was not previously signed. A secure system was designed under the assumption that factoring is hard, or a more general assumption that claw-free trapdoor permutations exist [15]. Recently, Bellare and Micali [1] have shown how to construct a secure signature system based on the assumption that trapdoor one-way permutations exist; this matches the original suggestion of Diffie and Hellman, but this time the system has a proof of security.

As one can see in the above history, the entire research in provably secure signatures was directed towards designing secure signature schemes based on the trapdoor property. In [15] the trapdoor property was even included as part of the definition of a signature scheme. Furthermore, there is no system which was proved secure even under a very weak attack and is based on a one-way function (even a special one) which is not trapdoor.

In this work, we present a *one-way based secure signature scheme*, a system based on the existence of 1-1 one-way functions; the system makes use of the primitive of universal one-way hash functions. Unlike the philosophy of previous systems, the signature algorithm is not based on a user's advantage of "knowing some trapdoor information" and can use a one-way key without a trapdoor. Nevertheless, the system is provably secure.

Two signature schemes not based on trapdoor functions were suggested previously, however neither of them was proved secure, even in the weakest sense of [15]: Fiat and Shamir [8] suggested a method on converting zero knowledge proofs for identification into efficient signatures schemes. Merkle [23] provided a pragmatic signature scheme based on any "encryption function". Our construction follows Merkle's paradigm and turns it into a provably secure scheme.

Besides the fact that we have reduced the sufficient conditions for having a secure signature, the scheme has other practical advantages. For example, even if the source of cryptographic security is a single instance of a one-way function (assumed to be hard to all participants) which is published by a central server (say, the NSA), still any individual in the community of users can base a signature-key of his own on this function instance and sign safely (a scenario similar to the identification scheme of [8]). Furthermore, concrete functions which are believed to be one-way require less time to compute than ones assumed to be trapdoor functions. Thus, basing a construction on one-way functions yields a more efficient one (unless the construction is very inefficient, which is not the case here).

Recently, Impagliazzo and Rudich [19] have shown a separation between the assumption that one-way functions exist and the assumption that trapdoor functions exist, at least in a certain restricted model. This provides even more incentive to try to weaken underlying assumptions, and to ascertain when the trapdoor property is needed.

Organization of the paper: In section 2 we present and construct universal one-way hash functions. In Section 3 we formally define secure signature schemes and in section 4 we construct our scheme and sketch the proof of its security. In section 5 we conclude at least some open problems.

2 Universal One-way Hash Functions

In this section we define universal one-way hash functions (UOWHF) and show how to construct them given any 1-1 one-way function. After we define UOWHF, in section 2.1 we show that composing families of UOWHF yields a family of UOWHF. In section 2.2

we show how to construct a family of universal one-way hash functions that compresses one bit. Combining this and the composition property allows us to construct a family for any input and output size that are polynomially related. Section 2.3 shows how to make UOWHF that have a more succinct representation and are more efficient to compute.

Let $\{n_{1_i}\}$ and $\{n_{0_i}\}$ be two increasing sequences such that for all i $n_{0_i} \leq n_{1_i}$, but $\exists q$, a polynomial, such that $q(n_{0_i}) \geq n_{1_i}$ (we say that these sequences are polynomially related). Let H_k be a collection of functions such that for all $h \in H_k$, $h : \{0,1\}^{n_{1_k}} \mapsto \{0,1\}^{n_{0_k}}$ and let $U = \bigcup_k H_k$. Let A be a probabilistic polynomial time algorithm (A is a *collision adversary*) that on input k outputs $x \in \{0,1\}^{n_{1_k}}$ which we call an *initial value*, then given a random $h \in H_k$ attempts to find $y \in \{0,1\}^{n_{1_k}}$ such that $h(x) = h(y)$ but $x \neq y$. In other words, after getting a hash function it tries to find a collision with the initial value.
Definition: Such a U is called a *family of universal one-way hash functions* if for all polynomials p and for all polynomial time probabilistic algorithms A the following holds for sufficiently large k.

1. If $x \in \{0,1\}^{n_{1_k}}$ is A's initial value, then $Prob[A(h,x) = y, h(x) = h(y), y \neq x] < 1/p(n_{1_k})$ where the probability is taken over all $h \in H_k$ and the random choices of A.
2. $\forall h \in H_k$ there is a description of h of length polynomial in n_{1_k}, such that given h's description and x, $h(x)$ is computable in polynomial time.
3. H_k is accessible : there exists an algorithm G such that G on input k generates uniformly at random a description of $h \in H_k$.

We note that we treat H_k as a collection of descriptions of functions; two different descriptions might correspond to the same function.

In this definition the collision adversary A is a (uniform) algorithm. We can alternatively define UOWHF where A is a polynomial sized circuit (the non-uniform case). In this case, all our results still hold, but we require the one-way functions that we use to be one-way in the non-uniform setting as well.

2.1 Composition of UOWHF

An important property of UOWHF families that we will make use of (and is probably important in applications) is that a composition of such families remains a family of UOWHF.

Let $H_1, H_2, \ldots H_l$ be families of functions such that $\forall i$ and $\forall h_i \in H_i$ $h_i : \{0,1\}^{n_i} \mapsto \{0,1\}^{n_{i-1}}$ and $n_i < n_{i+1}$. We call $H = \{h | h = h_1 \circ h_2 \circ \ldots \circ h_l\}$ an *l-composition* of $H_1, H_2, \ldots, H_l$. H is a multiset; if $h_1 \circ h_2 \circ \ldots \circ h_l = h'_1 \circ h'_2 \circ \ldots \circ h'_l$ for different $(h_1, h_2, \ldots, h_l)$ and $(h'_1, h'_2, \ldots, h'_l)$, both instances are members of H.

Lemma 2.1 *Let H be an l-composition as above. If there exists an algorithm A that produces an initial value x and when given a uniformly random $h \in H$ $Prob[A(h,x) = y, h(x) = h(y), y \neq x] > \epsilon$, then there exists an $1 \leq i \leq l$ and an algorithm A' such that*

- *A' produces an initial value $x_i \in \{0,1\}^{n_i}$*
- *then on input $h_i \in H_i$ tries to find a y_i that collides with x_i.*

- *$Prob[A'(h_i, x_i) = y_i, h(x_i) = h(y_i), y_i \neq x_i] > \epsilon/l$ where the probabilities are taken over $h_i \in H_i$ and A''s random choices.*
- *The running time of A' is polynomially related to that of A.*

Proof: Suppose that such an A exists and suppose that A's initial value is x and then A is given $h = h_1 \circ h_2 \circ \ldots \circ h_l$. Whenever A succeeds in finding a y such that $h(y) = h(x)$ but $y \neq x$ there is the first $1 \leq i \leq l$ where the composition of the h_j's on x and y becomes equal, i.e. $h_{i+1} \circ \ldots \circ h_l(y) \neq h_{i+1} \circ \ldots \circ h_l(x)$ but $h_i \circ \ldots \circ h_l(y) = h_i \circ \ldots \circ h_l(x)$. Hence, by the pigeon hole principle, there must exist an $1 \leq i \leq l$ where this occurs in at least $1/l$ of the cases. For that i, if $A(h, x) = y$, then

$$Prob[(h_{i+1} \circ \ldots \circ h_l(y) \neq h_{i+1} \circ \ldots \circ h_l(x)) \wedge (h_i \circ \ldots \circ h_l(y) = h_i \circ \ldots \circ h_l(x))] > \epsilon/l$$

where the probability is over uniformly random $h = h_1 \circ h_2 \circ \ldots \circ h_l \in H$ and A's random choices. This i would be the one for which we can "break" H_i (i.e. find collisions).

We can now define an algorithm A' that first produces an initial x_i, then given a randomly chosen $h_i \in H_i$ tries to find y_i that collides with x_i :

1. Given A's initial value x.
2. Choose at random $h_{i+1} \in H_{i+1}, h_{i+2} \in H_{i+2}, \ldots h_l \in H_l$ and output $x_i = h_{i+1} \circ \ldots \circ h_l(x)$.
3. on input h_i, choose at random $h_1 \in H_1, \ldots, h_{i-1} \in H_{i-1}$
4. run algorithm A on input $h = h_1 \circ \ldots \circ h_{i-1} \circ h_i \circ h_{i+1} \ldots h_l$.
5. if the output of A is y output $y_i = h_{i+1} \circ \ldots \circ h_l(y)$.

All the functions $h_1, h_2, \ldots, h_l$ are chosen uniformly at random from their respective families $H_1, H_2, \ldots H_l$, and thus h is uniformly distributed in H. Therefore the distribution on inputs that the simulated A sees is identical to the usual one and

$$Prob[(h_{i+1} \circ \ldots \circ h_l(y) \neq h_{i+1} \circ \ldots \circ h_l(x)) \wedge (h_i \circ \ldots \circ h_l(y) = h_i \circ \ldots \circ h_l(x))] > \epsilon/l.$$

Therefore $Prob[\wedge h(x_i) = h(y_i) \wedge y_i \neq x_i] > \epsilon/l$. Note that the running time of A' is the running time of A plus some polynomial amount of work. □

Remark: The composition lemma motivates the separation of choosing x from that of h in our definition of UOWHF families. If we had defined it differently, with x being part of the input chosen at random uniformly, the lemma wouldn't have been true. Counter-examples can be constructed similarly to ones in [11]. Note that in step 1 of algorithm A' the x_i is not necessarily chosen uniformly at random over all $\{0,1\}^{n_i}$.

Lemma 2.1 is a key component in the proof of the next theorem: Let $\{n_{0_i}\}, \{n_{1_i}\}, \{n_{2_i}\}, \ldots$ be a sequence of increasing sequences, let $U_1, U_2, \ldots$ be a sequence of families of UOWHF such that $U_i = \bigcup_k H_{i,k}$ where $\forall h \in H_{i,k}$ $h : \{0,1\}^{n_{i_k}} \mapsto \{0,1\}^{n_{(i-1)_k}}$. We also require that the U_i's be *simultaneously hard*: for every polynomial p and every probabilistic polynomial-time algorithm A, there is a K such that $\forall k > K$ and for all $i \geq 1$, A cannot succeed in finding collisions in $H_{i,k}$ with probability greater than $\frac{1}{p(n_{i_k})}$. Let $l : N \mapsto N$ be a polynomial computable function such that $q(n_{0_k}) > n_{l(k)_k}$ for some polynomial q. Let H_k be the $l(k)$-composition of $H_{1,k}, H_{2,k}, \ldots H_{l(k),k}$ and let $U = \bigcup_k H_k$.

Theorem 1 *U is a family of UOWHF.*

Proof: It is easy to see that H_k is accessible, and that given $h \in H_k$ computing $h(x)$ can be done in polynomial time. As for the hardness of finding collisions, assume the contrary, i.e. there exists an algorithm A such that for every polynomial p. for infinite k's, the probability that A succeeds in finding collisions for H_k is larger than $1/p(k)$.

Lemma 2.1 and the fact the U_i's are simultaneously hard imply that otherwise one of the U_i's would not be a family of UOWHF. □

2.2 Compressing One Bit

Theorem 1 (the composition theorem) tells us that in order to construct any family of UOWHF it suffices to have a construction for a family of UOWHF that compresses one bit, i.e that $h : \{0,1\}^k \mapsto \{0,1\}^{k-1}$ $\forall h \in H_k$. In this subsection we show how to compress one bit, assuming that a 1-1 one-way function is available. The construction is achieved by composing a universal hash function with the one-way function. We start by assuming that the one-way function is a permutation, then relax this condition to a 1-1 one-way function. A *one-way permutation* is a 1-1 length preserving function f that is polynomial-time computable, but for any polynomial p and any probabilistic polynomial time algorithm A (*an inversion adversary*) and sufficiently large k, $Prob\,[A(f(x)) = x] \leq 1/p(k)$ where the probability is taken uniformly over all $x \in \{0,1\}^k$ and A's internal random choices.

We will first assume that we have a one-way permutation f.

The source of the compression will be a family of strongly universal hash functions. Carter and Wegman [34, 4] defined *strongly universal*$_2$ functions as a family of functions G where $g : C \mapsto B$ for all $g \in G$ if for every pair of inputs (a_1, a_2) and pair of outputs (b_1, b_2), the number of functions that map a_1 to b_1 and a_2 to b_2 is $|G|/|B|^2$. If $g \in G$ is chosen uniformly then the the values of $g(x)$ and $g(y)$ are independent and uniformly distributed in B for any $x, y \in C$,

Collision Accessibility property: We require another property from the strongly universal$_2$ family G. Given that $g(x) = g(y)$ it is possible to generate in polynomial time a function $g \in G$ such that $g(x) = g(y)$ with equal probability over all functions in G which obey the restriction $g(x) = g(y)$.

For an example of such a family, consider the lines in the finite field $GF[2^k]$, $ax + b$ with the last bit chopped for the compression. $G_k = \{g_{a,b} | g_{a,b}(x) = chop(ax + b), a, b \in GF[2^k]\}$ where all the computation are in $GF[2^k]$ and $chop : \{0,1\}^k \mapsto \{0,1\}^{k-1}$ simply chops the last bit.

Define $H_k = \{h = g \circ f | g \in G_k\}$, where G_k is a strongly universal$_2$ family which has the collision accessibility property. The number of bits required to specify a member of H_k is the number needed to specify a member of G_k. In our example this is $2k$.

Lemma 2.2 *$U = \bigcup_k H_k$ is a UOWHF family.*

proof: We will show that if U is not a UOWHF family then we have an algorithm to invert a randomly chosen f on a random input, i.e. given a randomly chosen $f(w)$ we apply a randomized reduction from a collision adversary to an inversion adversary which will succeed in finding w with non-negligible probability.

Suppose that A is an algorithm that on a length k produces an initial $x \in \{0,1\}^k$ and when given $h \in H_k$ $Prob[A(x,h) = y \wedge h(x) = h(y) \wedge y \neq x] \geq \epsilon$ when the probability is over all $h \in H_k$ and A's random choices. We can define an algorithm A' so that given a random $z = f(w)$ tries to find w:

1. run A to produce x
2. choose a random $g \in G$ such that z and $f(x)$ collide, i.e., $g(f(x)) = g(z)$.
3. run A on input $h = g \circ f$ and x, let the output be y.

Step 2 is easy to perform (by choosing a random element to which g maps $f(x)$); the construction gives a 2-1 function and exactly one element collides with $f(x)$. Hence, if step 3 is successful, i.e. $h(x) = h(y)$ but $y \neq x$, then $f(y) = z$ and $y = w$.
Claim: if w was chosen at random, then the above procedure chooses h at random from H_k.
Proof: We can define a partition of H according to which element collides with $f(x)$. All parts are of equal size (by the property of strongly universal$_2$ hash functions). Since w was chosen at random and f is a permutation, a random part was chosen uniformly. Step 2 chooses with equal probability an h from that part. Hence, the whole procedure defines a random choice of h.

Since h is chosen uniformly at random, step 3 of the algorithm succeeds with probability at least ϵ, by assumption. Hence, the inversion succeeds with probability at least ϵ.$\Box$

This lemma in conjunction with theorem 1 and the fact that one-way functions over strings of polynomially related sizes are simultaneously hard yield the following:

Theorem 2 *If one-way permutations exist, then for any two increasing sequences* $\{n_{1_i}\}$ *and* $\{n_{0_i}\}$ *that are polynomially related there exists a family of UOWHF* $U = \bigcup_k H_k$ *such that* $h : \{0,1\}^{n_{1_k}} \mapsto \{0,1\}^{n_{0_k}}$ *for* $h \in H_k$.

The number of bits required to specify $h \in H_k$ using our example is $\sum_{i=n_{0_k}+1}^{n_{1_k}} 2i = n_{1_k}(n_{1_k} - 1) - n_{0_k}(n_{0_k} - 1)$.

Suppose we only have a 1-1 one-way function, i.e. unlike a one-way permutation it is not necessarily length preserving. It can be shown that the construction we gave for one-way permutation still works. We only have to use strongly universal functions over the range and define the chopping to be all but the last $k-1$ bits.

2.3 Efficient and Succinct UOWHF

Suppose we want to have public fingerprint, i.e. we have large files, stored in an insecure area, that we wish to hash down to a compact size which fits in the secure area. A hash function whose description is as long as the file would not help much; storing it in the secure area would require as much space as storing the file itself. Hence we develop in this section constructions for UOWHF families which have a more succinct representation.

Furthermore, these functions can be computed more efficiently then the ones of the previous section.

The first observation is that at each stage we can compress more than one bit. If, instead of using a 2-1 strongly universal hash function we used in the previous section, we use a t-1 strongly universal hashing, then the probability that step 3 of the algorithm suggested in the proof of lemma 2 produces the pre-image we want will be ϵ/t. Hence, as long as t is polynomial in the length, we have a polynomial algorithm to break f. If we compress a logarithmic number of bits, then t is polynomial. The total number of bits to specify a compression from n_1 to n_2 is therefore ${n_1}^2/\log n_1$. There is a similar reduction in the number of stages. (Furthermore, if we have a 1-1 one-way function whose inversion is hard for algorithms that run in time $O(2^{\delta k})$, then we can compress as many as δk bits at each stage.)

A much greater saving can be achieved by employing a block hashing technique similar to the one in [34]. Let m be large enough so that inverting f on m bits is infeasible. Divide the input into blocks of size $2m$. Let H be a UOWHF family whose members are $\{0,1\}^{2m} \mapsto \{0,1\}^m$. Let H' be the family you get by applying the same hash function that compresses from $2m$ to m bits on every block of the input. The members of H' are functions $\{0,1\}^{n_1} \mapsto \{0,1\}^{n_1/2}$. Breaking it requires breaking the $\{0,1\}^{2m} \mapsto \{0,1\}^m$ function on one of the blocks. Hence, when the number of blocks is polynomial in m (as we assume here), breaking H' means that at least one of the blocks has $1/poly(m)$ chance of being inverted and this can be used as an algorithm for breaking H. The number of bits needed to specify a member in H' is the same number needed to specify one in H, i.e., m^2. To further hash the file, we will use a new UOWHF family that again halves the number of bits. To hash from n_1 bits to n_0 bits we will continue inductively and will need to compose $log(n_1 - n_0)$ such functions. By the composition theorem this is still a UOWHF family. Thus to specify a function that compresses from n_1 bits to n_0 bits requires $O(m^2 \log n_1)$ bits. An additional advantage is that the one-way permutation (1-1 functions) has to be computed only on inputs of size m and the time complexity is essentially linear in n_1.

3 One-way based Signature Scheme

In this section we give our definition of a signature scheme (based on [15, 1]) and its security.

3.1 Definition of a Signature Scheme

A signature scheme includes the following components:

1. A *security parameter* k which determines the size of keys, messages and other resources; all sizes and algorithms are polynomial in k.

2. A *message space* MS; we allow all messages of a given size polynomial in k

3. A *key component* which includes a *key space* $KS(k)$, a family from which keys are being drawn and a *generation algorithm* KAL which chooses random keys.

4. A *signature bound* SB, a polynomial representing a bound on the number of messages signed; any polynomial should work.

5. A *system state* s which represents the state of the system; there are an initial state and execution states.

6. A *signing algorithm* SAL which is given a message, a system state, and a key, generates a signature and updates the system's state.

7. A *verification algorithm* VAL which is given a message, a signature and a system's state, checks the validity of the signature.

A signature system is a distributed system in which each user is a polynomial-time machine which initiates its instance of the signature scheme.

3.2 Security of a Signature Scheme

The most general attack on a signature scheme ([15]) has two phases. First, it allows a polynomial-time adversary F (a *forger*) to use the signature algorithm in an adaptive fashion, getting signatures to polynomially many plaintexts of its choice. Next, the attack has an existential nature, i.e., the forger itself has to come up with a valid signature of a new message of its choice, in which case we say that it was successful. A scheme is *p-forgeable* if for a polynomial p there is a forger F which for infinitely many k's, succeeds in the attack with probability larger than $1/p(k)$, where the probability is taken over the random choices of keys by KAL, the choices of the signatures by SAL, and the coin flips of F itself. We say that a system is *secure* if it is not p-forgeable for any polynomial p.

4 The Signature Scheme

Here we present our one-way based scheme: its components, algorithms, and security proof.

4.1 Background: Tagging System- "One-Time Signature"

The starting point of our system is the Diffie-Lamport *tagging system* [21]; both the system of Bellare and Micali [2] and the practical system of Merkle [23] which motivated us were based on it. The suggestion of [21] is to make public a one-way function f and *a window*, which is an ordered pair of values $< f(x_0), f(x_1) >$, for randomly chosen x_0, x_1 in the function domain. The user, then, is committed to the window and later on when it sends a bit b, it is done by publishing a tag x_b, an operation we call *opening half a window.* We say that the other half of the window remains *unused.*

The construction can be extended to tag a message of length m bits, by initially publishing (committing) to a *row of windows* $[< f(x_0^i), f(x_1^i) >, i = 1, \ldots, m]$ and then opening the halves corresponding to the bits of the message. Since f is one-way, only the committed user can open a tag, and no one else can tag a different message unless it can invert a random value of f, furthermore, anyone can verify tags; in this sense the system resembles a signature scheme. However, the drawback is that the size of the initial commitment limits the number of bits which can be tagged; in this sense it is not a signature scheme, which requires that once the initial key is placed in the public-key directory the system should be active "polynomially forever".

A tagging system provides what we may call a *one-time signature* (analogous to a finite one-time pad), which cannot be extended beyond a given length. Merkle gave a more space-efficient tagging system based on a tree structure [25].

4.2 An Overview

Here we present the simplest version of our system; in section 4.5 we suggest other versions.

The general strategy of the system is to extend the tagging system, enhancing it with the capability of "regenerating rows of windows". The system is represented as a linked list; a system's state is a list consisting of nodes. Each node is associated with a message, i.e., it tags that message. The node is also connected to its successor in the list; i.e., it tags the successor node as well.

The node N_i contains three data fields: h_i a universal one-way hash function, and two rows of windows rm_i and rs_i; the first will tag the next message M_{i+1} while the second one will tag the successor node in the list, N_{i+1}. The set of one-way permutations (1-1 functions) used in the tag encryption and in the hashing will be called the one-way function f. It is publicly known, or is otherwise produced by each user.

4.3 The System's Algorithms

Next we sketch the algorithms and the dynamic behavior of the system. The system has an initial state (state 0) in which a user deposits an initial (root) node N_0 in the public directory.

In a typical situation the system is in state s_{i-1} where there is a list of $i-1$ nodes and the *last-node* N_{i-1} is unused. The connection between nodes will be explained in the following sketch of the signature and the verification algorithms.

SAL: Each message signing changes the state of the system, the list grows by a node which becomes the new last-node. At state s_{i-1} the user sends a message M_i and tags it using the row rm_{i-1}. Furthermore, a new node N_i is generated by algorithm KAL: $N_i =< h_i, rm_i, rs_i >$ where its components are chosen at random: h_i is a random element of the UOWHF family based on f, and the rows are encryption by f of random tag values.

In order to link the new node into the list, the user has to tag the new node by its predecessor. Notice that the new node as a string of random bits is larger than the tagging capabilities of the row rs_{i-1} which was given this tagging task! Here is where the one-way hash is needed in a non-trivial way. The algorithm first computes the hash value of the new node by evaluating $n_i = h_{i-1}(N_i)$, then the smaller string n_i is tagged by opening the corresponding half-windows in rs_{i-1}. This defines a signature on M_i and a new valid state of the system s_i.

VAL: Verification of the validity of a message can be done by checking the tagging of the message M_i by rm_{i-1} and testing the validity of the system's state by checking that the tagging of $n_j = h_{j-1}(N_j)$ is a valid one, namely, it was done by a proper opening of rs_{j-1} for all $j = 1, \ldots, i-1$. This is done all the way to the root and if all checks are valid the user accepts the signature.

Remark: In practice, once old messages and signatures are out of date, a user can keep on-line only the relevant and previously verified suffix of the list; this saves time and space. With respect to a forger, though, the worst assumption is that it has access to the entire history at any time. In case not all users are following all message transmissions, a user can have a separate system dedicated to each other user, or a practical tree-like system can be used (to be described later).

Lemma 4.1 *The key-generation, signing, and verification algorithms are polynomial-time.*

4.4 Security of the Signature Scheme

The security of our scheme is based on a randomized reduction from a forgery to an inversion adversary of the one-way function f; it translates an existential attack to an attack which tries to invert a random element in the range of the function. The following lemma describes how a forger may be successful.

Lemma 4.2 *Assume a forger is successful in its attack after $i = poly(k)$ signature steps, then either (1) a half-window unused by the signature algorithm is opened, or (2) a node not generated by the signature algorithm (SAL) is tagged by a predecessor node generated by SAL.*

Given a successful forger F, we will use the lemma to generate an inversion adversary to the underlying one-way function f. Assume we are given a signature scheme, and assume a forger can successfully attack with a non-negligible success probability $\epsilon = \epsilon(k)$. Using the lemma above we know that either case (1) or (2) occurs with probability $\epsilon/2$ for infinitely many k's, (we say that the more frequent case dominates). If case (1) dominates then we can invert a random tag value; the reduction algorithm plays the role of SAL but plugs this value at a random location as half a window (this does not change the probability distribution of system states). With probability 1/2 the forger may ask to sign a message which contains a bit corresponding to this half a window, in which case we fail and stop. Otherwise, with probability inverse polynomial in the size of the history, the successful forger inverts this specific window. If case (2) dominates, the reduction algorithm plays the role of SAL again, generating random nodes, but at a random step it chooses a successor node first and then a hash function which will collide and, by theorem 2, will invert a random given element in the range of one of the one-way functions used in the hashing with probability inverse polynomial in the size of the history. (This construction does not change the probability distribution of system states.) Thus a successful forgery implies a successful inversion of the underlying one-way function f with probability polynomially related to ϵ, a contradiction. We summarize the above:

Theorem 3 *If 1-1 one-way functions exist, then the one-way based signature scheme described above is secure.*

4.5 Efficiency

Suggestions for improving the efficiency from [15, 9, 24, 1] are applicable to our scheme as well. Instead of a linked list we can arrange the one-time signatures in a d-way tree, where the parent tags all its d children. If k is the value of the security parameter and t messages have been signed so far, then the length of a signature in such a system is $O(k^2 \log_d t)$.

Using pseudo-random functions of [10] we can make our scheme "memoryless", in the sense that the signer need not remember the messages that he has signed or even their number. In particular many (authorized) signers can exist simultaneously without the need to coordinate their signatures. This is done by using the techniques of Levin and Goldreich [9] which were also used by [15, 1].

5 Conclusions

The significance of the this work is in identifying a "good" definition for one-way hash functions. We have seen that the definition is strong enough to implement construction such a public fingerprints and digital signature. On the other hand we have seen how any 1-1 one-way function can support the construction of such families. Very recently Rompel [32] has extended our work and showed how to construct a UOWHF from any one-way function.

It would be interesting to see more efficient constructions, perhaps based on more restrictive assumptions. One such construction was given by Impagliazzo and Naor in [18]. based of on the hardness of random subset sum problems of certain dimensions.

6 Acknowledgments

We would like to thank Manuel Blum, Benny Chor, Cynthia Dwork, Amos Fiat, Oded Goldreich, Stuart Haber, Ralph Merkle, and Silvio Micali for discussions and helpful comments.

References

[1] Bellare M. and S. Micali *How to Sign Given any Trapdoor Function* , Proc. 20th Annual Symposium on the Theory of Computing, Chicago, Il, 1988, pp. 32-42.

[2] Blum M. and S. Micali *How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits*, SIAM Journal on Computing, V. 13, No. 4, Nov. 84, pp. 850-864.

[3] G. Brassard, C. Crepeau and M. Yung, *All NP Can Be Proved in Perfect Zero-Knowledge in Constant Rounds*, ICALP 1989, to appear.

[4] J. L. Carter and M. N. Wegman, *Universal Classes of Hash Functions*, Journal of Computer and System Sciences 18 (1979), pp. 143-154.

[5] I. B. Damgard, *Collision Free Hash Functions and Public Key Signature Schemes* , Eurocrypt, 1987.

[6] W. Diffie and M. Hellman, *New Directions in Cryptography* , IEEE Trans. on Information Theory, vol. IT-22, 6 (1976), pp. 644-654.

[7] S. Even, O. Goldreich, and S. Micali, *On-line/Off-line Digital Signatures*, Proc. Advances in Cryptology – Crypto '89, Springer Verlag, pp. 263–275, 1990.

[8] A. Fiat and A. Shamir, *How to Prove Yourself: Practical Solutions to Identification Problems and Signature Problems*, Procôf Crypto 86, pp. 186-194.

[9] O. Goldreich, *Two Remarks Concerning the GMR Signature Scheme*, Procôf Crypto 86, pp. 104-110.

[10] O. Goldreich, S. Goldwasser and S. Micali, *How to Construct Random Functions* Journal of the ACM (1986).

[11] O. Goldreich, H. Krawczyk and M. Luby, *On the existence of Pseudorandom Generators*, Proceedings of the 29th Symposium on the Foundation of Computer Science , 1988, pp. 12-24.

[12] S. Goldreich, S. Micali and A. Wigderson, *Proofs that Yields Nothing But their Validity, and a Methodology of Cryptographic Protocol Design*, Proceedings of the 27th Symposium on the Foundation of Computer Science, 1986, pp. 174-187.

[13] S. Goldwasser and S. Micali, *Probabilistic Encryption*, J. Com. Sys. Sci. 28 (1984), pp. 270-299.

[14] S. Goldwasser, S. Micali and C. Rackoff, *The Knowledge Complexity of Interactive Proof-Systems*, Proc. 17th Annual Symposium on the Theory of Computing, Providence RI,, 1985, pp. 291-304.

[15] S. Goldwasser, S. Micali and R. Rivest, *A secure digital signature scheme* , Siam Journal on Computing, Vol. 17, 2 (1988), pp. 281-308.

[16] S. Goldwasser, S. Micali and A. C. Yao, *Strong signature schemes* , Proc. 15th Annual Symposium on the Theory of Computing, Boston, Ma, 1983, pp. 431-439.

[17] R. Impagliazzo, L. Levin and M. Luby, *Pseudo-random Generation given from a One-way Function*, STOC 89.

[18] R. Impagliazzo and M. Naor, *Efficient Cryptographic Schemes Provably as Secure as Subset Sum*, Proc. of the 30th Symp. on Foundations of Computer Science, 1989, pp. 236–241. Full version: Technical Report CS93-12, Weizmann Institute.

[19] R. Impagliazzo and S. Rudich, *Limits on the Provable Consequences of One-way Permutations*, These Proc.

[20] R. Impagliazzo and M. Yung, *Direct Minimum-Knowledge Computations* , Proc. of Crypto 87, Springer Verlag.

[21] L. Lamport, *Constructing digital signatures from one-way functions*, SRI intl. CSL-98, October 1979.

[22] L. Levin, *One-way Functions and Pseudorandom Generators*, Proc. 17th Annual Symposium on the Theory of Computing, 1985, pp. 363-365.

[23] R. Merkle, *A Digital Signature based on Conventional Encryption Function*, Crypto 1987, Springer Verlag.

[24] R. Merkle, *Secrecy, Authentication and Public Key Systems*, Ph.D. Thesis (1982), UMI Research Press, Ann Arbor, Michigan.

[25] R. Merkle, *A Certified Digital Signature*, Manuscript 1979.

[26] R. Merkle and M. Hellman, *Hiding Information and Signature in Trapdoor Knapsack*, IEEE Trans. on Information Theory, vol. IT-24, 5 (1978), pp. 525-530.

[27] M. Naor, *Bit commitment using pseudo-randomness* Proc. of Crypto 89.

[28] M. O. Rabin *digitalized signatures* , in Foundation of Secure Computation, Academic Press, R.A. DeMillo, D. Dobkin, A. Jones and R. Lipton, eds., Academic Press, 1977.

[29] M. O. Rabin *Fingerprinting by Random Polynomials* , Harvard University, TR-15-81, 1981.

[30] M. O. Rabin *Digital Signatures and Public Key Functions as Intractable as Factoring*, Technical Memo TM-212, Lab. for Computer Science, MIT, 1979.

[31] R. Rivest, A. Shamir and L. Adleman, *A Method for Obtaining Digital Signature and Public Key Cryptosystems*, Comm. of ACM, 21 (1978), pp. 120-126.

[32] J. Rompel, *One-way functions are necessary and sufficent for secure signatures*, Proc. 22nd ACM Annual Symposium on the Theory of Computing, 1990, pp. 387–394.

[33] A. Shamir, *On the Generation of Cryptographically Strong Pseudo-Random Number Sequences*, ACM Trans. Comput. Sys., 1 (1983), pp. 38-44.

[34] M. N. Wegman and J. L. Carter, *New Hash Functions and Their Use in Authentication and Set Equality*, Journal of Computer and System Sciences 22, pp. 265-279 (1981).

[35] A. C. Yao, *Theory and Applications of Trapdoor functions*, Proceedings of the 23th Symposium on the Foundation of Computer Science, 1982, pp. 80-91.