COLLISION FREE HASH FUNCTIONS AND PUBLIC KEY SIGNATURE SCHEMES

Ivan Bjerre Damgård[1]

Aarhus University, Inst. of Math.
Ny Munkegade,
DK 8000 Aarhus C,
Denmark

Abstract

In this paper, we present a construction of hash functions. These functions are collision free in the sense that under some cryptographic assumption, it is provably hard for an enemy to find collisions. Assumptions that would be sufficient are the hardness of factoring, of discrete log, or the (possibly) more general assumption about the existence of claw free sets of permutations.

The ability of a hash function to improve security and speed of a signature scheme is discussed: for example, we can combine the RSA-system with a collision free hash function based on factoring to get a scheme which is more efficient and much more secure.

Also, the effect of combining the Goldwasser-Micali-Rivest signature scheme with one of our functions is studied. In the factoring based implementation of the scheme using a k-bit modulus, the signing process can be speeded up by a factor roughly equal to $k \cdot O(\log_2(k))$, while the signature checking process will be faster by a factor of $O(\log_2(k))$.

1. Introduction

One of the most fascinating features of public key cryptography is the notion of digital signatures. However, for many of the so far proposed schemes a proof of security does not (yet) exist, or they have been shown to be breakable under sufficiently strong attacks. Moreover, a practical implementation of a signature scheme is often made very difficult by the complexity of the algorithms needed in the system. These problems suggest the use of a hash function, some suitable transformation which is applied to a message before signing it. In particular, we would like to mention the following:

- with a block oriented signature algorithm where messages are longer than a block, it is not safe to sign messages block by block: an enemy could remove blocks from a signed message or insert blocks of his choice into a message before it was signed. Thus, some transformation must be used to make a signature depend on all parts of a message.

[1]This research was supported by the Danish Natural Science Research Council.

- if the message space has some algebraic structure, and the signing algorithm behaves too nicely with respect to this structure, the system can be vulnerable to a chosen message attack (see e.g. [De]). A hash function can be used to destroy this algebraic structure.
- usually, the output of the hash function is much shorter than the input, so if the signature algorithm is slower than the hash function used, a considerable amount of time can be gained in an implementation of the scheme.

The need for hash functions has been realized before ([De], [DP]), and several attempts have been made to construct such functions using fx. DES or RSA as building blocks. However, none of these suggestions have been proved to be secure, and several of the proposals using DES have been proven insecure ([Wi], [Co]). Other variations of hash functions have also been proposed [We]. But the use of these functions, like pseudo random functions [GGM], require that sender and receiver share a secret key. They are therefore suitable for authentication purposes, but do not fit into the scenario of a public key signature scheme. We would like to have publicly known hash functions that are easy for all users to compute.

2. Construction of Collision Free Hash Functions

In stead of considering just one hash function, we will consider families of them, in order to make a complexity theoretic treatment possible. Any member of such a family will have a value of *security parameter* attached to it. A number of things, such as the overall security of a system using the hash function, will depend on it. For any security parameter value k, we choose a finite alphabet Σ_k with $|\Sigma_k| = r_k$. A hash function with security parameter value k will be a map $h: M_k \rightarrow A_k$, where M_k is the set of all finite words over Σ_k, and A_k is some finite set. Note that when no confusion is possible, the subscript k's will sometimes be dropped.

The most basic demand to a good hash function is that it should be computationally infeasible for an enemy to find collisions, i.e. different messages hashing to the same value. There are a number of possible interpretations of what ''computationally infeasible'' means. In this paper we will choose circuit complexity to describe it, because it seems best suited in a cryptographic setting (cf. [BM], p.857). But note that other computational models, e.g. Turing machines, would allow our results to be proven in the same way, and would only result in a change in the intractability assumptions we make later.

Throughout this paper, a problem will be said to be *computationally infeasible* to solve, if the following is satisfied:

- Let $\{C_k\}$ be a boolean circuit family of polynomially bounded size. Let ε_k be the fraction of the instances of size k which are solved by C_k. Then, as a function of k, ε_k vanishes faster than any polynomial fraction.

Loosely speaking, a problem is hard in above sense if no polynomial size circuit can solve more than a negligible fraction of the instances. Whenever this definition is used in this paper, k will be interpreted as a security parameter value. ''Polynomial time'' will always mean "computable by a polynomial (in k) size circuit family. Finally, a probability is "negligible" if it vanishes faster than any polynomial fraction as a function of the security parameter.

Definition 2.1
A *family of collision free hash functions* is a set of hash functions with the following properties:

- There is a probabilistic polynomial time algorithm, which on input a value of security parameter selects uniformly and randomly a member of the family with the given value attached.
- All functions in the family are computable in polynomial time.
- The problem of finding $x \neq y$ such that $h(x)=h(y)$ for a given h in the family is computationally infeasible to solve $\square$

A member of a collision free family is also called collision free. If h is collision free, then h is also one-way in the following sense:

Lemma 2.2
Given a collision free family, and a member $h: M \rightarrow A$. For any finite $W \subseteq M$ with $|W| - |A|$ a nonnegligible fraction of $|A|$, it is computationally infeasible given $h(w)$, where $w \in W$, to find any $w' \in W$ with $h(w') = h(w)$, for more than a negligible fraction of the w in W.

Proof.
Assume the lemma is false. Choose $w \in W$ at random and compute $h(w)$. By assumption, we can efficiently find w' such that $h(w)=h(w')$. By assumption on $|W|$, the probability that $w \neq w'$ is nonnegligible, which means that this procedure creates a collision for h with nonnegligible probability $\square$

It is important to note that since A might be negligibly small compared to W, the above lemma does not imply that it is hard to invert h on almost all elements in A: even though the set of elements $w \in W$ for which it is easy to invert h on $h(w)$ is negligibly small compared to W, their images might constitute a large fraction of A! It turns out that this last form of one way property is important in connection with chosen message attacks on signature schemes. Each hash function therefore has to be checked for this one way property. More details can be found in [Da].

In [GMR], the notion of claw free pairs of trapdoor permutations are introduced. We will use a generalisation of this idea, without the trapdoor property.

Definition 2.3
A *claw free family of permutations* is a family of sets of permutations with the following properties:

- Each set S in the family has a security parameter value attached to it, and there exists a function $g: \mathbb{N} \rightarrow \mathbb{N}$ such that when S has security parameter k, then $|S| = g(k)$.
- All members of a set S in the family have the same domain.
- There is a probabilistic polynomial time algorithm, which on input a security parameter value selects randomly and uniformly a member of the family with the given value attached.
- For each set $S = \{f_0, \ldots, f_{r-1}\}$, and each $x \in \mathit{domain}(f_0)$, it is easy to compute $f_i(x)$ for all $i = 0 \cdots r-1$, but it is computationally infeasible to create *a claw*, i.e. find x and y such that for some $i \neq j$, $f_i(x) = f_j(y)$ □

Assuming the existence of claw free permutations, we can construct collision free hash functions. First some notation:

Let an alphabet Σ with cardinality r and a finite word m over Σ be given. We now let $[m]$ denote a prefix free encoding of m over Σ. The choice of a particular encoding is not important to the results given here, except for the fact that it is possible to encode efficiently so that the length of $[m]$ is a linear function of the length of m. In binary, for example, 1 could be encoded as 11, 0 as 00, and all encodings could be terminated with 01. This is important because a short encoding will make the mechanisms shown here more efficient. In fact, theoretical results show that the encoding can be chosen such that the length of m is almost equal to the length of $[m]$. In this case, however, the encoding process itself might become inefficient. Further details on prefixfree encodings can be found in [BP].

Now, if $\{f_0, \ldots, f_{r-1}\}$ is a set of permutations, all with domain D, we define

$$f_{[m]}(I) = f_{m_1}(f_{m_2}(\cdots f_{m_s}(I)\cdots)),$$

where $I \in D$, $[m] = m_1 m_2 \cdots m_s$, and the letters in Σ are denoted by the numbers $0, \ldots, r-1$. A similar construction is used in [GMR] with $r=2$.

Theorem 2.4
The family F constructed below is a collision free family of hash functions.

Let P be a family of claw free permutations. For each value of the security parameter k, we let Σ_k be the alphabet of cardinality $r_k = g(k)$ given by $\Sigma_k = \{0, 1, \ldots, r_k - 1\}$. For each set $S = \{f_0, \ldots, f_{r_k-1}\} \in$ P and each $I \in \mathit{domain}(f_0)$, we define a member h of F with security parameter k by:

$$h(m) = f_{[m]}(I),$$

for all $m \in M_k$.

Proof

Assume for contradiction that F is not collision free. This means that for any $h \in$ F, we can efficiently find $m \neq m'$, such that

$$f_{[m]}(I) = f_{[m']}(I),$$

where $[m]$ and $[m']$ have lengths s and s', respectively. Note that since m and m' are assumed to be produced by a polynomial size circuit, both s and s' can be at most polynomial in k. If $m_1 \neq m'_1$, we have a claw for the set S. If $m_1 = m'_1$, the fact that the f's are injective implies that

$$f_{m_2}(\cdots f_{m_s}(I)\cdots) = f_{m'_2}(\cdots f_{m'_{s'}}(I)\cdots).$$

The same argument now applies again, and since the encoding used is prefix free, this process must stop with the creation of a claw $\square$

If we let T denote the time needed to evaluate one of the permutations used in the construction, it is clear that the time needed to compute h on a message of length L is $O(TL)$.

The motivation for working with sets of claw free permutations rather than just pairs (as in [GMR]) is now clear: any binary message can be seen as a word over a larger alphabet by treating s-bit chunks of it as symbols in an alphabet with 2^s elements. Thus if we have claw free sets with 2^s elements, the message can be hashed by processing s bits of it in stead of 1 at time.

2.1. Examples of Claw Free Permutations

We first give a construction of a claw free family of permutations under the assumption that factoring is hard.

Choose any polynomially bounded function $g : \mathbb{N} \rightarrow \mathbb{N}$. For each value of the security parameter k, we define the permutation set size to be $g(k)$, for short denoted by r_k. Now, let $n = p_1 p_2 \cdots p_t$, where all the p's are k bit prime numbers equivalent to 3 modulo 4, and where t is the smallest integer such that $2^{t-1} \geq r_k$. The set of integers of this form is denoted H_k. For each $n \in H_k$ we shall construct a set of claw free permutations with security parameter k.

For each a prime to n, define

$$J(a) = ((\frac{a}{p_1}), (\frac{a}{p_2}), \cdots, (\frac{a}{p_t})).$$

$QR(n)$ will be the set of quadratic residues mod n. Clearly,

$$QR(n) = \{a \mid J(a) = (1,1,\cdots,1)\}.$$

The set of all ± 1 t-tuples form a group under pointwise multiplication. Let G_t denote this group modulo the subgroup generated by $(-1,-1,\ldots,-1)$. Clearly J induces a surjective homomorphism $\phi_n : \mathbf{Z}_n^* \to G_t$. Choose a set of r_k elements in $\mathbf{Z}_n^*$, $A = \{a_0, \ldots, a_{r_k}-1\}$ such that $|\phi_n(A)| = |A|$. This is clearly possible by choice of t. A is called *an injective set of numbers*, whenever it satisfies this condition. We can now define our set of permutations $\{f_0^{(n)}, f_1^{(n)}, \ldots, f_{r-1}^{(n)}\}$ to be the set of permutations of $QR(n)$ given by

$$f_i^{(n)}(x) = (a_i x)^2 \bmod n, \quad \text{for } x \in QR(n) \text{ and } i = 0, \ldots, r-1.$$

To prove that finding claws is as hard as factoring, it turns out to be essential that the a_i's (and not just their squares) are made public. It might be argued that this could endanger the factorization of n, since checking whether a set is injective requires knowledge of the factors, and since the set size grows exponentially with the number of factors. To prove that such release of an injective set is not dangerous, we need the following series of technical, but elementary lemmas:

Lemma 2.5.
Let G be a finite abelian group of exponent 2. Let $S = \{g_1, \ldots, g_s\} \subseteq G$. Then $\langle S \rangle$, the subgroup generated by S, has order at most 2^s, and equality occurs exactly when no g_i can expressed as a product of the others.

Proof.
Trivial from the fact that all g_i's have order 1 or 2 □

Lemma 2.6
Let G be a finite abelian group of exponent 2, $|G| = 2^s$. The probability that a randomly chosen subset S of G of cardinality s generates G is

$$p_s = \frac{2^s-1}{2^s} \cdot \frac{2^{s-1}-1}{2^{s-1}} \cdot \ldots \cdot \frac{2-1}{2}.$$

Moreover,

$$p_s \to p_\infty = \frac{1}{3} - \frac{1}{3 \cdot 7} + \frac{1}{3 \cdot 7 \cdot 15} - \cdots \approx 0.289 \text{ for } s \to \infty$$

Proof.
Let $S = \{g_1, \ldots, g_s\}$. For all $1 \le i \le s$ we have that

$$Prob(g_{i+1} \notin \langle g_1, \ldots, g_i \rangle) = \frac{2^s - 2^i}{2^s} = \frac{2^{s-i}-1}{2^{s-i}},$$

provided $\langle g_1, \ldots, g_i \rangle$ has maximal size, by Lemma 2.5. Thus we get

$$Prob(|\langle g_1, \ldots, g_{i+1}\rangle| = 2^{i+1}) = \frac{2^{s-i}-1}{2^{s-i}} \cdot Prob(|\langle g_1, \ldots, g_i\rangle| = 2^i).$$

This proves the first statement. The second follows from a long series of tedious and rather trivial manipulations with the expression for p_s. The reader will be spared the details □

Lemma 2.7

Consider a probabilistic polynomial size circuit family that on inputs an integer $n \in H_k$ and an injective set for n, factors n with probability p_k. For any such circuit family and any $\varepsilon > 0$ there exists another probabilistic polynomial size circuit family that factors n with probability q_k, such that $|p_k - q_k| < \varepsilon$.

Proof.

The idea is to guess an injective set and to feed this and n to the factoring circuit we are given. It suffices to consider the case where the set size r_k equals 2^{t-1} (smaller sets are easier to guess). Choose $t-1$ numbers at random from Z_n^*, call them $a_1, \ldots, a_{t-1}$. Under ϕ_n, they correspond to $t-1$ randomly chosen elements in G_t. By Lemma 2.6, if we form the set

$$A = \{x \in Z_n^* \mid x = \prod_i a_i^{\nu_i},\ \nu_i = 0 \text{ or } 1\},$$

then the probability that $\phi_n(A) = G_t$ (and hence that A is an injective set) is at least p_∞. Since r_k is at most polynomial in k, the process of choosing A can be completed by a polynomial size circuit. By going through this procedure s times, we obtain for the probability of success, q_k, that

$$p_k \geq q_k \geq p_k(1-(1-p_\infty)^s).$$

Since p_∞ is independent of k, this proves the lemma.

Theorem 2.8

The family consisting of all sets of permutations constructed as above is a claw free family of permutations, provided factoring integers in $\{H_k\}$ is computationally infeasible.

Proof.

Using well known algorithms and the Chinese remainder theorem, the p's and the a's for each n can easily be selected. Further, each permutation can be evaluated with two modular multiplications. Thus it suffices to prove that finding claws is as hard as factoring. Suppose a claw for the set $\{f_0^{(n)}, \ldots, f_{r-1}^{(n)}\}$ has been found. So for some $i \neq j$ we have:

$$f_i^{(n)}(x) = f_j^{(n)}(y)$$

or

$$(a_i x - a_j y)\cdot(a_i x + a_j y) \equiv 0 \bmod n.$$

By choice of the a's and basic properties of the Jacobi symbol, this means that

$GCD(a_i x + a_j y, n)$ will produce a nontrivial factor of n. By Lemma 2.7, we can now efficiently factor n if claws can be found efficiently $\square$

With this construction of claw free sets, it will often be possible to improve performance of our hash functions by using larger sets, rather than fx. pairs. One should be aware, though, that with this construction a larger set requires more prime factors and hence a longer modulus - note that a modulus with many shorter prime factors will not be safe enough because of factoring algorithms like Lentra's elliptic curve method [St], whoose running time depends mostly on the size of the smallest prime factor.

There is, however, a slight variation on this construction which does not have this drawback. It was inspired by a construction of "blobs", which was presented in [BC]. In this case, let n be a modulus with exactly 2 k-bit prime factors equivalent to 3 mod 4. Next, choose a set $A = \{a_1, \ldots, a_{r-1}\} \subseteq QR(n)$. Define a set of permutations $\{f_1^{(n)}, \ldots, f_{r-1}^{(n)}\}$ by

$$f_i^{(n)}(x) = a_i x^2, \text{ for } x \in QR(n).$$

It is now trivial to prove:

Lemma 2.9

Suppose an adversary can compute a claw for $f_i^{(n)}$ and $f_j^{(n)}$ as defined above. Then he can compute a square root of $a_i \cdot a_j^{-1}$.

The practical advantage of this is that the set A can be enlarged without changing the modulus, so that one can use as many a's as one is willing to store (or generate pseudo randomly). Asymptotically, since we can only use a polynomial size A, this means that this variation is faster than the previous construction by a factor of $O(\log_2(k))$. It does have a drawback, which is only of theoretical significance, however: with the previous construction, we could prove that for *any* choice of injective set, knowledge of a claw implied knowledge of a factor of n. In the new construction, it is conceivable that an adversary could know some square roots of quotients of elements in A just "by chance", and not as a result of possessing an algorithm for computing square roots of randomly chosen elements in $QR(n)$. We may assume, however, that the probability of this is negligible in practice: if the elements of A are chosen at random, then the probability that some polynomial size circuit can find their square roots is negligible, unless square roots can be computed in random polynomial time. More details can be found in [Da].

The final example is based on the hardness of discrete log, and shows that claw free permutations need not be trapdoor. It is of theoretical significance for that reason, but of no great practical use, since one permutation evaluation needs a full modular exponentiation. Choose a large prime p, a generator g of Z_p^*, and a set $A = \{a_0, \ldots, a_{r-1}\} \subseteq Z_p^*$. Define

$$f_i(x) = a_i g^x, \text{ for } x \in Z_p^*, \text{ and } 0 \le i \le r-1.$$

In the same way as before, it is easy to prove that to find claws, one must be able to find the discrete log base g of quotients of elements in A.

3. Combining a Signature Scheme and a Hash Function

The model we shall use is the following: we have a signature scheme with

- message space A
- signature space B
- and a signature algorithm Δ,

which for any message $a \in A$ produces as output a signature $\Delta(S, a) \in B$, using the secret key S.

Δ could be probabilistic, and its output could depend on the number of messages previously signed, or how they were signed in which case the signature scheme is called non-memoryless. Since memorylessness is not relevant to the results proved below, the dependence on random inputs or previously signed messages has been omitted from the notation. Finally we have

- a verification predicate Γ,

which for any message a and signature s produces as output a boolean value $\Gamma(P, a, s)$ using the public key P. $\Gamma(P, a, s) = \text{TRUE}$ if and only if s is a valid signature for a using the secret key matching P.

This scheme is referred to as *the original scheme.* A full model of a signature scheme would also have to include an algorithm that generates matching pairs of public and secret keys, but this will not be important in this context.

As usual, a number of things such as the length of the keys, the running time of Δ and Γ, the overall security of the scheme depend on the value of a security parameter k.

We now combine this scheme with a hash function $h: M \rightarrow A$. A message is signed by computing $\Delta(S, h(m))$ and signatures are checked by computing $\Gamma(P, h(m), \Delta(S, h(m)))$. This scheme will be called *the combined scheme.*

When combining with a collision free hash function, we always assume for simplicity that the signature scheme and the hash function have the same value of security parameter.

Following [GMR], a signature scheme is called *existentially forgeable* if under some attack, an enemy can forge the signature of at least one message. This message cannot necessarily be chosen by the enemy.

A *chosen message attack* is an attack where the enemy can choose messages to be signed by the legitimate signer before trying to forge a signature. If he is also allowed to choose such messages *during* the process of forgery, we speak of an *adaptive chosen message attack.*

It is of course essential to be able to compare the security of the two schemes. A first result in this direction is:

Theorem 3.1

Suppose that the combined scheme is at least existentially forgeable under a chosen message attack, and that the hash function used is collision free. Then the original scheme can be existentially forged under a chosen message attack.

Proof.

Assume for contradiction that we have an algorithm Φ that forges at least one signature in the combined scheme after receiving signatures of messages of its choice. Assume also that we can mount a chosen message attack on the original scheme. While running Φ, we will be asked for signatures of messages in some subset N of M. They are easily found by computing the set $h(N)$ and using the chosen message attack to obtain $\Delta(S, h(N))$. With nonnegligible probability, Φ will produce as output a message $m \notin N$ and a valid signature for m. If $h(m) \in h(N)$ we have found a collision for h, but by assumption on h this can only occur with negligible probability. We may therefore assume that $h(m) \notin h(N)$ which means that we have forged a new signature in the original scheme □

Note that the proof goes through in the same way if the attack is adaptive. Therefore "adaptive chosen message attack" could substituted for "chosen message attack" in the theorem.

This result has two useful implications:

- If the original scheme is not existentially forgeable, then neither is the combined scheme.
- If the combined scheme has any kind of weakness, then the original scheme was already existentially forgeable.

The last fact states that the use of the hash function does not introduce any "completely new" weaknesses. This may make a security analysis of the combined scheme easier, but unfortunately it does not rule out the possibility that the combined scheme is much weaker than the original scheme! This will be illustrated by an example later.

The next result, however, shows that at least a large class of attacks directed against the original scheme are prevented by the use of a hash function with the right property:

Proposition 3.2

Suppose the hash function used is one way in the sense that it is hard to invert on a randomly chosen element in A (c.f. the discussion following Lemma 2.1). Consider an algorithm for forging signatures which is directed against *the original scheme*, i.e. the hash function used is not part of the input to the algorithm. No such algorithm which satisfies i) or ii) below will be of any use in attacking the combined scheme.

i) The algorithm only works under a chosen message attack, and will during the attack ask for signatures of messages uniformly distributed in A.

ii) The algorithm can only forge signatures for a small (i.e. polynomial) number of messages, which are non-chosen and uniformly distributed over the message space (where the distribution is taken over all choices of messages to sign made by the signer and all choices made by the enemy).

Proof

i): by the one way property, even a chosen message attack on the combined scheme does not allow a chosen message attack on the original scheme: Since h is not part of the input to the forgery algorithm, any procedure which could implement a chosen message attack on the original scheme based on a chosen message attack on the combined scheme, would in fact be an inversion algorithm for h. ii): Call the set of messages for which the algorithm has forged signatures X. Since X is in fact a random variable, then to use these signatures, the enemy must have an algorithm that on input any set of messages Y with $|Y| = |X|$ efficiently produces an element $m \in M$ with $h(m) \in Y$. But this would enable him to invert h on a randomly chosen element with non negligible probability $\square$

Let us see whether the functions constucted in the previous section have the one way property required in the proposition. As far as the functions based on factoring are concerned the answer, curiously, is yes if discrete log is hard! A simple example: suppose we construct a hash function based on the pair of permutations

$$f_0(x) = x^2 \text{ and } f_1(x) = ax^2$$

where, as usual, all computations are modulo an appropriately chosen modulus n. We now chose a random $I \in SQ(n)$ and define $h(w) = f_{[w]}(I)$ But with these permutations, it is easy to see that in fact

$$h(w) = a^{[w]} \cdot I^{2^{l(w)}},$$

where $l(w)$ is the length in bits of $[w]$. Thus, even if an enemy when presented with an element in $Im(h)$ could guess the length of a preimage, he would still have to solve a discrete log problem base a. Moreover, the set of discrete log problems that arise in this way certainly constitute a non negligible fraction of all possible discrete log problems base a.

As far as hashfunctions based on claw free permutations in general are concerned, it seems to be hard to give actual proof that the one way property is always satisfied. It is possible, however, to give heuristic arguments suggesting that such functions will hit any element in their image with almost equal probability. This and the collision freeness would imply the one way property (more details on this can be found in [Da]).

Assumptions i) and ii) in the proposition above seem to be reasonable models of the multiplicative attacks on RSA mentioned in [De]. Thus a collision free function would prevent these. But let us stress once again that the proposition does not talk about attacks directed against the *combined* scheme. Therefore an analysis of this combination is needed in each concrete case.

As an example of these problems, consider a combined scheme with RSA as the original scheme, and a hash function based on factoring as described in Section 2.

If the modulus used for the hash function is chosen *independently* of the RSA-modulus, then the following assumption seems reasonable:

If there exists an algorithm that forges RSA signatures under condition 1) below, then there exists an equally efficient algorithm that forges signatures under condition 2).

1) All messages signed are of the form $h(m)$, and h and m are known to the forgery algorithm.

2) All messages are chosen at random from $\mathrm{Im}(h)$.

In other words, as far as forging RSA-signatures is concerned, the hash function might as well be a random function. If this is indeed true, then by Proposition 3.2 the only way to break the combined scheme is to totally break RSA under a non chosen message attack (i.e. forge the signature of any message with nonnegligible probability). It therefore seems that this combination is much more secure than plain RSA.

A word of warning, however: if the two moduli used are not independent, then the above assumption can be seriously violated. For example, if for convenience we use the same modulus for the hash function as for the RSA, then by an attack due to Bert den Boer [Bo], the combined scheme can in fact be broken. The attack relies heavily on the fact that the multiplicative structure used in computing the hash function is exactly the same as the one used in computing signatures. It therefore does not generalise to the case where the moduli are independently chosen.

It is worth noting that in addition to being more secure, the combination with RSA has a number of practical advantages:

- Since modular arithmetic is used for both the hash function and RSA, the combination can be implemented without using more hardware than is needed for basic RSA.
- Using the most efficient construction mentioned in Section 2 will yield a scheme which is more efficient than plain RSA on long messages. For example, if one is willing to store a set of 512 constant coefficients, then the combined scheme will be 9 times faster than the original one.
- Signatures have length 1 RSA block, independent of the message length.

4. Speeding up the Goldwasser Micali Rivest Signature Scheme

In [GMR], the Goldwasser-Micali-Rivest (GMR) signature scheme is introduced and is proven to be not existentially forgeable, even under an adaptive chosen message attack. The basic building blocks in the scheme are two pairs of claw free trapdoor permutations, (f_0, f_1) and (g_0, g_1). To sign a message a, the following is done:

1) A random value R is chosen.
2) A value α is computed using the trapdoor information for the g-permutations such that $g_{[a]}(\alpha) = R$.
3) R is authenticated using the f-permutations.

The signature for a now consists of α, R, and the authentication for R. It is pointed out that the scheme is most practical when messages have length much greater than the value of security parameter used, since in this case signatures are short compared to the messages. Clearly, if α is computed in a straightforward way by inverting the g-permutations involved one by one, then the time needed to compute a signature for a message of length L is $O(LT_{inv})$, where T_{inv} is the time needed to invert one permutation. Moreover, if (as proposed in [GMR]) we use the factoring based construction of claw free pairs of permutations, then T_{inv} is greater than the time for one permutation evaluation by a factor of $\log_2(n)$, where n is the modulus used.

Using the constructions from Section 2, we can improve the efficiency of this scheme substantially:

1) It is trivial to check that the GMR scheme will work equally well with any size of sets of claw free permutations. Therefore we can use the most efficient construction from Section 2 in place of both the f- and the g-permutations. This will speed up both the signing and the signature checking process by a factor of $O(\log_2(\log_2(n)))$.
2) By combining with a collision free hash function based on factoring, we can avoid most of the permutation inversions. It is easy to check that this will speed up the signing process further by a factor of about $\log_2(n)$ for long messages. Moreover, by Theorem 2.4 and 3.1, the combined scheme is as secure as the original one.

Other methods for speeding up the GMR scheme have been proposed, both by Goldreich [Go] and Goldwasser, Micali and Rivest in the full version of their paper [GMR2]. They both apply only to the signing process and achieve a slightly smaller improvement than our method (the factor gained is $\log_2(n)$). They are also fundamentally different: the method from [Go] only applies to the factoring based implementation of claw free permutations, while our method is potentially useful with any implementation. The method from [GMR2] will not allow the use of non trapdoor permutations, while the construction of a hash function only uses the claw free property.

5. Conclusion

We have seen that hash functions can be useful, both in improving the security and the efficiency of signature schemes. We have also seen that if factoring or discrete log is hard, provably secure and reasonably efficient hash functions can be constructed.

References

[BC] Brassard and Crepeau: Non transitive transfer of confidence: a perfect zero-knowledge interactive protocol for SAT and beyond. Proc. of FOCS 86.

[Da] Damgård: Application of claw free functions in cryptography; unconditional protection in cryptographic protocols. Phd Thesis, in preparation.

[De] Denning: Digital signatures with RSA and other public key cryptosystems. CACM, v.27, nr 4, 1984, pp.388-392.

[GMR] Goldwasser, Micali and Rivest: A "paradoxical" solution to the signature problem. Proc. of 25th FOCS, 1984, pp.441-448.

[BP] Berstel and Perrin: Theory of codes, Academic Press, 1985.

[DP] Davis and Price: The application of digital signatures based on public key cryptosystems. Proc. of 5th Int. CompCom, 1980, pp.525-530.

[Wi] Winternitz: Producing a one-way hash function from DES. Proc. of Crypto 83, pp.203-208.

[Co] Coppersmith: Another birthday attack. Proc. of Crypto 85.

[St] Stephens: Lentras factoring method based on elliptic curves. Proc. of Crypto 85.

[WC] Wegman and Carter: New hash functions. JCSI, v.22, 1981.

[GGM] Goldreich, Goldwasser and Micali: How to construct random functions. MIT.LCS.TM-244, 1983.

[BM] Blum and Micali: How to generate cryptographically strong sequences of pseudorandom bits. SIAM J. of Computing, v.13, 1984.

[Go] Goldreich: Two remarks concerning the GMR-signature scheme. Proc. of Crypto 86.

[GMR2] Goldwasser, Micali and Rivest: A digital signature scheme secure against adaptive chosen message attacks. To appear.

[Bo] Bert den Boer: private communication.