Journal of the Nigerian Association of Mathematical Physics
Volume **11 (November 2007), 467 - 474**

Incremental cryptography and security of public hash functions

Tola John Odule
Department of Mathematical Sciences ,
Olabisi Onabanjo University
Ago-Iwoye, Ogun Stat, Nigeria

Abstract

An investigation of incremental algorithms for crytographic functions was initiated. The problem, for collision-free hashing, is to design a scheme for which there exists an efficient "update" algorithm: this algorithm is given the hash function H, the hash h = H(M) of message M and the "replacement request" (j, m), and outputs the hash H(M(j, m)) of the modified message. Ideally, the update time should depend only on the block size b and the underlying security parameter k, and not on the length of the message.

Keywords: Cryptography, incrementality, public hash function, security, efficiency, random access machine, Turing machine, substitution attack.

1.0 Introduction

Incrementality is fundamentally a practical concern because it is a measure of efficiency. Clearly, an (ideal) incremental scheme is a win over a standard one, as message sizes get larger. The practical concern is what is the cross-over point: if incrementality only helps for messages longer than one is ever likely to get, one is not inclined to use the incremental scheme. The cross over point for the schemes presented here is low enough to make them interesting.

This is achieved in the constructions presented here. Finally, all security results are stated and analysed exactly (as opposed to asymptotically) and strive for the best possible reductions. The special case of this hash function in which the number of blocks n is a constant was presented and analyzed by Chaum, Heijst and Pfitzmann [5]. Brands [3] provided a proof of security for $n = poly(k)$.

Note that the hash functions discussed here are ones of public description. That is, the description of the function is provided to the adversary trying to find collisions. This is unlike the hash functions used in applications like fingerprinting, where the description of the function is not available to the collision-finder!

1.1 Fundamental Consideration

The notion of basic security makes an assumption. Namely, that the signer is in a setting where the integrity of messages and signatures which he is updating is assured. That is, when a signer applies the update algorithm to update M and its signature σ, he is confident that this data has not been tampered with since he created it. This is reflected in the fact that adversary's attack on the update algorithm consists of pointing to a past (authentic) message/signature pair.

This is the right assumption in the majority of applications of digital signatures. For example, in the case where one is sending the same message to many parties except with different headers, One signs one copy and update to obtain the rest. But one keeps the original copy and its signature on one's machine--when one updates one knows the original is authentic.

But there are some situations in which one might want an even stronger form of security. For example, suppose one is remote editing a file residing on an insecure machine, and at any time a virus, which would tamper with the data, could hit the machine. For efficiency one is incrementally signing the file every time a change is made to it. But when the update algorithm is run, one can't be sure the data is still authentic. (It is impractical to verify authenticity before updating because verification takes time depending on n and the whole point of incrementality is to update the signature quick).

A new notion of security is hereby formalized under substitution attacks appropriate to the above setting. It is then shown that substitution attacks can be used to break the above hash-and-sign scheme when the hash function is a discrete log based one. This is interesting in two ways--it illustrates the strength of the new attacks, and it shows that a "standard" construction (namely hash-and-sign) can become insecure in a new setting!

1.2 Hash Functions

A *hash function H* is a transformation that takes an input *m* and returns a fixed-size string, which is called the hash value *h* (that is, $h = H(m)$). Hash functions with just this property have a variety of general computational uses, but when employed in cryptography, the hash functions are usually chosen to have some additional properties as follows: the input can be of any length; the output has a fixed length; $H(x)$ is relatively easy to compute for any given x; $H(x)$ is one-way and $H(x)$ is collision-free.

A *one-way function* [10] is a mathematical function that is significantly easier to compute in one direction (the forward direction) than in the opposite direction (the inverse direction). It might be possible, for example, to compute the function in the forward direction in seconds but to compute its inverse could take months or years, if at all possible. A *trapdoor one-way function* is a one-way function for which the inverse direction is easy given a certain piece of information (the trapdoor), but difficult otherwise.

Damgård and Merkle [9] greatly influenced cryptographic hash function design by defining a hash function in terms of what is called a *compression function*. Given a compression function, a hash function can be defined by repeated applications of the compression function until the entire message has been processed. A message of arbitrary length is broken into blocks whose length depends on the compression function, and "padded" (for security reasons) so the size of the message is a multiple of the block size. The blocks are then processed sequentially, taking as input the result of the hash so far and the current message block, with the final output being the hash value for the message (Figure 1.2).

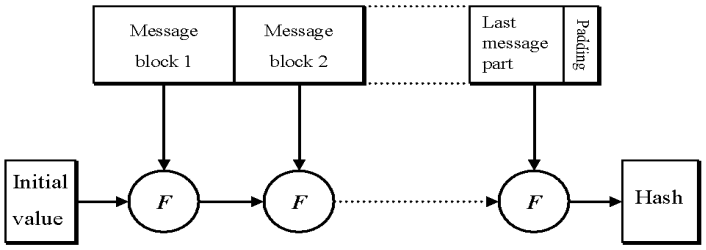

Figure 1.2: Damgård/Merkle iterative structures for hash functions (*F* is a compression function)

2.0 Families of hash functions

The usual definitions of hash families [7] are hereby extended to allow independent consideration of the security parameter, the block size and the number of blocks. These parameters are denoted *k; b; n,* respectively. Below the string *H* is (the description of) a particular hash function.

***Definition* 2.0**

A family of hash functions is specified by a pair IH $= (HGen, Heval)$ of algorithms.

- The *PPT generator HGen* takes as input 1^k; 1^b; 1^n and outputs a string *H*.
- The polynomial time *hash evaluation algorithm HEval* takes *H* and a message [1] $M \in B_b^n$ and outputs a *k*-bit string called the hash of *M* under *H*.

When the family $(HGen, HEval)$ is clear from the context, *H* shall be identified with $HEval(H,.)$ and regard it as a map of B_b^n to $\{0,1\}^k$. In particular *H(M)* shall be written for $HEval(H, M)$.

2.1 Incrementality

The following definition says that an update algorithm *lncH* is one that can turn the hash of *M* into the hash of [2]$M\langle j.m\rangle$.

***Definition* 2.1**

Let IH = (HGen, Heval) specify a family of has functions. We say that lncH is an update algorithm for, with running time $T(\cdot,\cdot,\cdot)$ if

$$\forall k, b, n,\ \forall H \in \left[HGen\left(1^k, 1^b, 1^n\right)\right] \forall j \in \{1, \cdots, n\},\ \forall m \in B_b$$

if h = HEval(H,M) *then it is the case* l*ncH*$(H,M,h(j,m))$ *halts in T*(k,b,n) *steps with output equal to HEval*$(H,M\langle j,m\rangle)$.

The **lncH** – augment of **IH** = (*HGen, HEval*) is the triple IH^+ = (*IHGen, HEval, lncH*). It should be noted that this definition makes no requirement on the running time *T* in *lncH*. So, in particular, an update algorithm can just run can just run *HEval*$(H,M\langle j,m\rangle)$ to compute its output. This is not intended to be excluded--it is a legitimate update algorithm. But of course an update It should be emphasized that an "ideal" update algorithm is one whose running time does not depend on *n*. Such an algorithm would random access a small number of relevant memory blocks (this is where the RAM model is needed as opposed to the Turing machine model) and do some quick computation before writing the output back to memory.

3.0 Security

A proof of the security of IH^+ takes the following form. Given a collision finder *A* for $IH^+(k;^1k; n)$, construct a discrete log finder *B* for [4]*G(k)*. Now suppose *A* succeeds in (t,ε)*-breaking* $IH^+(k; k; n)$**.** The question considered is for what values of t',ε' the constructed algorithm *B* succeeds in (t',ε')*- breaking G(k)*.

It should be noted that previous works [5, 3] have only discussed asymptotic security, where one sets $n = n(k)$ to some fixed polynomial in *k*, regards $t,\varepsilon, t',\varepsilon'$ as functions of *k*, assumes *t*, ε are polynomial and non-negligible, respectively, and then shows that t',ε' are also polynomial, and non-negligible, respectively. But for practice it is important to know exactly how the resources and achievements of *B* compare to those of *A*, so that one may know what size to choose for the prime *p* and what adversaries one can tolerate with a specific security parameter. Moreover, it is important to strive for the tightest possible reduction, because this means that the same "[5]security" can be obtained with a smaller value of the security parameter, meaning greater efficiency. Thus one wants the effort and success of *B* should be as close to those of *A* as possible.

In this light let's look at the existing reductions to see what they achieve. The proof of [5] only applies to the case of $n = O(1)$ block messages, and in fact *t'* seems to grow exponentially with *n*, so that this reduction is not suitable for our purposes. Brands [3] proposes a reduction which removes the

restriction on *n* and achieves $t' = t + O(nK^3)$ and $t' = \in / n$. The running time of *B* here is essentially optimal: one must think of *t* as much larger than *n* or *k*, and additive terms like the $O(nk^3)$ correspond to overhead of *B* coming from simple and unavoidable arithmetic operations.

The loss in the success probability is more serious. Note that (particularly in this case) *n* may be very large. Thus even if *A* is successful with high probability, the above may only let one conclude that *B* is successful with low probability. The reduction has been improved to be essentially optimal. The current quality of the running time is also preserved, and thus achieve for *B* a success probability within a small constant factor of that of *A*.

The big-oh notation, both in the time as given above and in the following theorem, hides a constant which depends only on the underlying machine model and can be taken as small in a reasonable setting. Let *U* denote some oracle machine, which depends only on our proof and the given family. Although the statement of the theorem does not say anything about the "size" of *U*, the proof shows that it is "small," and this is important in practice. $\boldsymbol{IH}^+$ is the **lncH**-*augmentation* of ***IH*.**

***Theorem* 3.0**

There is an oracle machine U such that the following is true. Suppose collision-finder A succeeds in (t,ε)-breaking, $\boldsymbol{IH}^+(k; k; n)$*. Then discrete log finder* $B \underline{\underline{def}} \mathbf{U}^A$ *succeeds in* $(t', \in')$*-breaking* $G(k)$ *where* $t' = t + O(nK^3)$ *and* $\in' = \in / 2$.

Proof

The algorithm *U* is first described. Then it is argued that its running time is as claimed and finally that its success probability is as claimed. On inputs *p,g,x* algorithm B selects $r_1, \cdots, r_n \in (0,1)$ at random and $u_1, \cdots, u_n \in (0,1, \cdots, p-1)$ at random. For $i = 1, \cdots, n$ it sets $g_i = \begin{cases} g^{u_i}, if\ r_i = 0 \\ x^{u_i}, if\ r_i = 1 \end{cases}$ it sets $H = (p; g_1, \cdots, g_n)$. Now it invokes $A(H)$ and obtains distinct messages

$$M_1 = M_1[1], \ldots, M_1[n] \quad and \quad M_2 = M_2[1], \ldots, M_2[n] \qquad (1)$$

For *j* = 1, 2 it is now convenient to set $t_{j,i} = \langle M_j[i] \rangle$. Algorithm *B* sets $\alpha = \Sigma_{r_i = 1} u_i (t_{1,i} - t_{2,i})$, the arithmetic here being modulo p. If this quantity is 0 then *B* has failed, and it halts with no output. So assume it is non-zero. Now compute an inverse *b* of *a* mod *p* (i.e., $ba \equiv 1 \bmod p$. Such an inverse always exists since p is prime, and it can be found via Euclid's algorithm). *B* computes $\alpha = b.\Sigma_{r_i = 0} u_i (t_{2,i} - t_{1,i}) \bmod p$ and halts. *B* invokes *A* once. In addition it performs some arithmetic modulo *p* of which the dominant part is *O(n)* exponentiations. This accounts for the claimed running time. We now turn to justifying the claimed success probability.

Note that the distribution of $g_1, \ldots, g_n$ is uniform and independent and is the same as the distribution over these quantities that **HGen** would generate. So the messages found by B in Equation 1 are a collision, i.e., $H(M_1) = H(M_2)$ with probability at least ϵ. Now assuming they are a collision we have $\prod_{i=1}^{n} g_i^{t_{1,i}} = \prod_{i=1}^{n} g_i^{t_{2,i}}$. Using the definition of $g_1, \ldots, g_n$ and rearranging terms in the above we get $\prod_{r_i = 1} x^{u_i (t_{1,i} - t_{2,i})} = \prod_{r_i = 0} g^{u_i (t_{2,i} - t_{1,i})}$. Note that the left hand side is x^a. It is now claimed that with probability at least ½ we have $a \neq 0$. Given this, raise both sides of the above equation to the power *b* to get $x = x^{ab} = \prod_{r_i = 0} g^{b u_i (t_{2,i} - t_{1,i})} = g^{\alpha}$ showing that α is indeed $\mathbf{index}_g^G p\,(x)$. It remains to justify the claim. This is argued informally using the following technical fact.

■

3.1 Technical Fact

Let $a_1,\ldots,a_n$ be numbers with the property that $\Sigma_{i=1}^{n} a_i \neq 0$. Let $X_1,\ldots,X_n$ be independent random variables defined by $\Pr[X_i = a_i] = \Pr[X_i = 0] = 1/2$ for each $i = 1,\ldots,n$. Let $X = \Sigma_{i=1}^{n} X_i$. Then $\Pr[X \neq 0] \geq 1/2$. It is noted that the distribution on $g_1,\ldots,g_n$ is independent of $r_1,\ldots,r_n$. Thus the experiment may be thought of as the following game: Choose $g_1,\ldots,g_n$ at random and obtain the collision from *A*. Let $a_i = u_i(t_{1,i} - t_{2,i})$ for $i = 1,\ldots,n$. Now choose $r_1,\ldots,r_n$ at random and compute $\Sigma_{r_i=1} a_i$. Viewed this way it is seen that it is the same as the technical fact stated above.

3.2 Efficiency

Hashing an *n*-block message takes *n* exponentiations (equivalently, one multiplication per message bit) modulo a *O(k)*-bit prime. This is quite good for a number-theory based scheme. How does it compare with standard hash functions like MD5 or SHA? Let's fix k = 512. In hashing from scratch there is no comparison--MD5 on 512n bits is far better than *n* exponentiations. But assuming one is in a setting with frequent updates. With MD5 [13], there is no choice but to hash from scratch, while in this scheme one can use the update algorithm to update the hash in two exponentiations. Thus to compare the efficiency one should ask how large is *n* before the time to do two exponentiations of 512 bit numbers is less than the time to evaluate MD5 on a 512*n* bit string. A computation that yields a reasonable value. Note, however, that there are heuristics (based on vector-chain addition) to compute $\prod_{i=1}^{n} g_i^{\langle M[i] \rangle}$ faster than doing *n* modular exponentiations [2].

4.0 A practical version with small description size

The size of (the description of) the hash function in the previous section is *O(nk)* so that it depends on the message size, which we assume large. In practice this is too much. Here is a suggested way to reduce the size to *O(k)*. Let $f : \{0,1\}^k \rightarrow \{0,1\}^{O(k)}$ be the restriction of some "standard" hash function, such as MD5, to inputs of length *k*. We now set $g_i = f(i)$ to be the result of evaluating *f* at *i*. Now the description of the hash function is just the prime *p* and anyone can quickly compute $g_1,\cdots,g_n$ for himself. The loss in efficiency is negligible since the time for the arithmetic operations dwarfs the MD5 computation time.

Although such a construction must ultimately be viewed as heuristic, its security can be discussed by assuming *f* is a random function. Extending the proof of security given earlier to this setting is not difficult and it can be concluded (the following statement is informal) that the scheme just described satisfies Theorem 3.0 in the random oracle model. As discussed by [1], although this approach (namely prove security in a random oracle model and then instantiate the random oracle with a standard hash function) does not yield provable security, it provides a better guarantee than purely heuristic design, and protocols designed in this manner seem to be secure in practice.

The hardness of discrete log implies, via [12], the existence of standard (i.e., non-incremental) signature schemes which can play the role of S^* in the above. Combining this with the results of Section 3.0 one has established the existence of an incremental signature scheme with short signatures given the hardness of the discrete log in groups of prime order. This construction however is not too practical because of the use of the result of [12]. For a practical version one could use El Gamal's scheme [8] or RSA in the role of S^* and the practical version of the hash function presented herein (cf. Section 4.0) in the role of ,.

The public file is large because the hash function has poly(*n*, *k*) size. But it isn't necessary that each user publishes a hash function. Rather, some (trusted) center can publish a single hash function for use by all users. Now, a user's public file is just that of the original non-incremental scheme, and this is poly(*k*).

4.1 The tree hash Function

The tree-hash scheme is probably the first thing that comes to mind when asked to find an incremental signature scheme. Assuming for simplicity that $b = k$ we recall that the scheme makes use of a standard (i.e., Not necessarily incremental) collision-free hash function $H : \{0,1\}^{2k} \to \{0,1\}^{k}$. The message is hashed by the binary tree construction. That is, in each stage, adjacent blocks are hashed together to yield a single block, halving the number of blocks per stage. In lg(*n*) stages we have the final hash value. This can be signed under the standard scheme.

Now suppose we store all the internal nodes of the tree: formally, include them in the signature. Now the hash can he incremented by just recomputing the tree nodes indicated by the path from the updated block to the root of the tree. The security needs again to be reconsidered because we allow the adversary to attack the update algorithm but some thought shows that the scheme satisfies the basic security requirement. But the signature is long--incrementality is at the cost of storing about twice as many bits as in the message. Thus while this scheme may be incremental under the formal definition presented here, it is too memory inefficient to be interesting in most applications. What is desired are schemes with short signatures.

4.2 A successful substitution attack

The strength of substitution attacks is hereby illustrated by showing how hash function can be broken. (In particular this means the scheme in question should not be used in applications like remote editing a file on a machine, which could be unexpectedly hit by a virus). The attack is interesting in illustrating how substitution attacks work. It is also interesting in illustrating how a "standard" construction like hash-and-sign, which is secure in the usual sense, fails to be secure in a new setting such as *remote computing*.

For simplicity assume that the messages consist of just one block $(n = 1)$: the attack easily generalizes to arbitrary *n*. The hash function is described by $(p : g)$ and reduces simply to $H\left(M = g^{(M)} = g^{1+M}\right)$, the operations being in G_p. Let Sk^* be the signing key under the standard scheme, so that the signature of *M* is $\sigma = (g^{1+M}, \sigma^*)$ where $\sigma^* \xleftarrow{R} Sig^*(Sk^*, g^{1+M})$

The adversary *F* begins with the simple signing request *A*. The reply she obtains has the form $\sigma_A = (h_A, \sigma_A^*)$ where $h_A = g^{1+A}$. Think of it as the signer having signed *A* and stored A, σ_A on the insecure medium. Here, set $M_1 = A$.

Now, *F* makes the incremental signing request $(B, \sigma_A, (1, C), 1)$. That is, on the insecure medium, she changes *A* to *B*, and asks the signer to substitute *C* for the first (and only) block of this message. According to the scheme being presented here, the signer first applies the hash update algorithm to update the hash: $h_F = h_A \cdot g^{-(1+B)} \cdot g^{1+C} = g^{1+A-B+C}$. Then he re-signs via $\sigma_F^* \xleftarrow{R} Sig^*(Sk^*, h_F)$. The reply to *F* is $\sigma_F = (h_F, \sigma_F^*)$.

What is important to note at this point is that what the signer really believes himself to have signed is *C*. Thus, the adversary can simply output $(A - B + C, \sigma_F)$ as a forgery. The verification algorithm will accept σ_F as the signature of $A - B + C$. But at this point the set of messages whose signatures have been legally obtained is $Legal = \{A, C\}$. For appropriate choices of B, C (it suffices that $B \notin \{A, C\}$ it is the case that $A - B + C \notin Legal$. Thus the adversary is successful, and the scheme is broken with probability one.

Notice that the attack did not find collisions in *H*, nor did it forge signatures under Sk^*. It is not known whether the attack applies to any instance of the hash-and-sign paradigm, but the above is sufficient to show hash-and-sign is not in general secure again**st substitution attack.**

5.0 Conclusion

A new notion of security is hereby formalized under substitution attacks appropriate to the above setting. It is has been shown that substitution attacks can be used to break the above hash-and-sign scheme when the hash function is a discrete log based one. This is interesting in two ways--it illustrates the strength of the new attacks, and it shows that a "standard" construction (namely hash-and-sign function) can become insecure in a new setting!

---

[1] B_b^n -is the space of *n*-block messages.

[2]Message consisting of *M* with block *j* replaced by *m*.

[3] Take $b = k$

[4]PrimeGen is a Probabilistic Polynomial Time (PPT) algorithm, which when fixed on input 1^k outputs a *k* +1-bit prime *p* identifying a group G_p of (prime order *p)*. *G*(k) = {*Gp*:*p* Є [*PrimeGen*(1^k)]}. Such roups have been used for cryptography by Croft and Harris [6], Schnorr [14], Chaum and Van Antwerpen [4].

[5]In keeping with the philosophy of an adaptive-chosen message attack [11], one must allow the adversary to obtain examples of signatures under the algorithm given by the hash function.

[*] For example the port of a traditional UNIX program. Its *main()* method is called right after the object is created and executes until the object is destroyed. Such an object must be moved to a remote node, along with all the threads that are executing within it.

References

[1] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. Proceedings of the First Annual Conference on Computer and Communications Security, ACM, 1993.

[2] J. Bos and M. Coster. Addition chain heuristics. Advances in Cryptology { Crypto 89 Proceedings, Lecture Notes in Computer Science Vol. 435, Springer-Verlag, G. Brassard, ed., 1989.

[3] S. Brands. An efficient off-line electronic cash system based on the representation problem. CWI Technical Report CS-R9323.

[4] D. Chaum and H. Van Antwerpen. Undeniable signatures. Advances in Cryptology - Crypto 89 Proceedings, Lecture Notes in Computer Science Vol. 435, Springer-Verlag, G. Brassard, ed., 1989.

[5] D. Chaum, E. Heijst and B. Pfitzmann. Cryptographically strong undeniable signatures, unconditionally secure for the signer. Advances in Cryptology - Crypto 91 Proceedings, Lecture Notes in Computer Science Vol. 576, Springer-Verlag, J. Feigenbaum, ed., 1991.

[6] Croft and Harris. Public-key cryptography and reusable shared secrets. In Cryptography and Coding, Clarendon Press, 1989.

[7] I. Damgard. Collision-free hash functions and public-key signature schemes. Advances in Cryptology Eurocrypt 87 Proceedings, Lecture Notes in Computer Science Vol. 304, Springer-Verlag, D. Chaum, ed., 1987.

[8] T. El Gamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Info. Theory, Vol. IT 31, 1985.

[9] I. Damgård. A Design principle for hash functions. Advances in Cryptology - Crypto 89, Springer-Verlag (1990), 416-427.

[10] O. Goldreich and L. Levin. A hard predicate for all one-way functions. Proceedings of the Twenty First Annual Symposium on the Theory of Computing, ACM, 1989.

[11] S. Goldwasser, S. Micali and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal of Computing, 17(2):281{308, April 1988.

[12] M. Naor and M. Yung. Universal One-Way Hash Functions and their Cryptographic Applications. Proceedings of the Twenty First Annual Symposium on the Theory of Computing, ACM, 1989.

[13] R. Rivest. The MD5 message-digest algorithm. IETF Network Working Group, RFC 1321, April 1992.

[14] C. Schnorr. Efficient identification and signatures for smart cards. Advances in Cryptology - Crypto 89 Proceedings, Lecture Notes in Computer Science Vol. 435, Springer-Verlag, G. Brassard, ed., 1989.