Appears in *Advances in Cryptology – Crypto 94 Proceedings*, Lecture Notes in Computer Science Vol. 839, Springer-Verlag, Y. Desmedt, ed., 1994.

Incremental Cryptography: The Case of Hashing and Signing

M. Bellare* O. Goldreich† S. Goldwasser‡

December 10, 1995

Abstract

We initiate the investigation of a new kind of efficiency for cryptographic transformations. The idea is that having once applied the transformation to some document M, the time to update the result upon modification of M should be "proportional" to the "amount of modification" done to M. Thereby one obtains much faster cryptographic primitives for environments where closely related documents are undergoing the same cryptographic transformations.

We provide some basic definitions enabling treatment of the new notion. We then exemplify our approach by suggesting incremental schemes for hashing and signing which are efficient according to our new measure.

* Advanced Networking Laboratory, IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA. e-mail: `mihir@watson.ibm.com`.

† Department of Applied Mathematics and Computer Science, Weizmann Institute of Science, Rehovot, Israel. e-mail: `oded@wisdom.weizmann.ac.il`. Research was partially supported by grant No. 92-00226 from the US-Israel Binational Science Foundation (BSF), Jerusalem, Israel.

‡ Department of Applied Mathematics and Computer Science, Weizmann Institute of Science, Rehovot, Israel, and MIT Laboratory for Computer Science, 545 Technology Square, Cambridge MA 02139, USA. e-mail: `shafi@wisdom.weizmann.ac.il`. Research was partially supported by NSF FAW grant CCR-9023313 and DARPA grant N00014-92-J-1799.

1 Introduction

We initiate an investigation of *incremental* algorithms for cryptographic functions. The idea, taking digital signatures as an example, is to have a signature which is easy to update upon modification of the underlying message. Thus, suppose you have signed message M to obtain signature σ. Now you make some change to M. For example, you might replace one block by another, insert a new block, or delete an old one. Let M' be the modified message. You want to update the signature σ into a signature σ' of M'. The time to update should be somehow proportional to the "amount of change" you have made in M to get M', and not the same as simply signing M' anew.

1.1 A wide range of applications

Incrementality is suitable for an environment in which the documents undergoing cryptographic transformations are altered versions of documents which have already undergone the same cryptographic transformations.

For example suppose you are sending the same message to many different users so that the text is essentially the same except for header information, and you want to sign each copy.

A second example is video traffic. Here one can take advantage of the well-known fact that successive video frames usually differ only slightly.

A third example is the use of authentication tags for virus protection. We imagine a user who has a PC and stores his files on a remote, possibly insecure host which could be attacked by a virus. The user authenticates the files so that he can detect their modification by a virus. When a user modifies his files, he must re-authenticate them, and an incremental scheme might be useful. (The type of authentication used for virus protection can vary. The most likely choice may be a private key authentication scheme such as a finger-print. But one can envisage applications where the user's signature has to be checked also by others and a full fledged digital signature is desirable).

In general it seems clear that incrementality is a nice property to have for any cryptographic primitive (eg. finger-printing, message authentication, digital signatures, hashing, encryption, etc.).

1.2 Problems considered in this abstract

It quickly becomes apparent that incrementality presents a large area of research. The goal of this (preliminary) abstract is to draw attention to this area, lay some basic definitions, and provide some examples. For simplicity we restrict the scope of the work presented here in two ways. First, we focus on just one update operation on the underlying message, namely the replacement of one block by another. Second, we limit the primitives we consider to digital signatures and the collision-free hashing primitive via which we approach it.

We view a message $M = M[1] \ldots M[n]$ as a sequence of b-bit blocks. Let $M\langle j, m\rangle$ denote M with $M[j]$ replaced by the b-bit string m. 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 M and the "replacement request" (j, m), and outputs the hash $H(M\langle j, m\rangle)$ of the modified message. Similarly, for signing, the update algorithm is given the signing key Sk, a message M, its signature σ, and the replacement request (j, m), and must produce the signature σ' of $M\langle j, m\rangle$. Ideally, in either case, 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.[1] A scheme is said to be ideally incremental if it possesses an ideal update algorithm.

In work in progress we address other operations on messages, like insertion or deletion of blocks, and we also expand the scope to consider more primitives, namely finger-printing and message authentication. See Section 5 for more information.

1.3 Incremental collision-free hashing

We pin-point an ideal incremental scheme for collision-free hashing which is based on the hardness of the discrete logarithm in groups of prime order. Hashing an n-block message (each block k bits long) to a k bit string takes n exponentiations, while updating takes two exponentiations; these exponentiations are modulo a $O(k)$-bit prime.

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 [7]. Brands [5] provided a proof of security for $n = \text{poly}(k)$. The observation that this hash function possesses an (ideal) update algorithm identifies for the first time a crucial use for it: we know no other collision-free hashing scheme that is ideally incremental.

We make an additional contribution by considering the "exact security" of the reduction via which security is proved, and presenting a new reduction which is essentially optimal. The motivation and impact of this result, as explained in Section 3.4.3, is practical: it enables a user to get the same level of security for a smaller value of the security parameter, leading to greater efficiency.

Note the hash functions we discuss 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 finger-printing, where the description of the function is not available to the collision-finder!

1.4 Incremental digital signatures

With an (ideal) incremental hash function available, an (ideal) incremental signature scheme is easily derived by a slight variation of the standard hash-and-sign construction. Namely, hash the message M with the incremental hash function to get h; sign h with some standard scheme to get σ^*; and use (h, σ^*) as the signature. (The variation is that the hash value must be included in the signature). To update, update the hash (fast by assumption) and then sign the new hash value from scratch (fast because we are signing a k-bit string).

When evaluating the security of this scheme (or any other) scheme one must be careful: the presence of the update algorithm entails new security considerations. In keeping with the philosophy of an adaptive chosen message attack [13], we must allow the adversary to obtain examples of signatures under the update algorithm. In general these could be differently distributed from signatures created by the signing algorithm, and the extra power could help the adversary.

We will show that the above scheme achieves what we call *basic* security. Here, in addition to being able to get examples of signatures from the signing algorithm, the adversary can point to any past message/signature pair and obtain the result of an arbitrary update on this pair.

[1] Some care must be taken in formalizing this since the update algorithm receives the entire nb-bit message as input and in a standard Turing machine model could not even read the relevant part in $\text{poly}(k, b)$ time. Thus we assume a RAM type computation model and in the formal definition allow the update time to depend on $\log n$.

1.5 Practical issues

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 I am ever likely to get, I am not inclined to use the incremental scheme. The cross over point for our schemes is low enough to make them interesting (cf. Section 3.4.4).

We prefer schemes with low memory requirements. Signatures of size proportional to the message, as permitted in theoretical definitions, are not really acceptable. Thus we want schemes with poly(k, b) size signatures independent of the number n of message blocks. (This consideration eliminates some trivial incremental schemes like the tree hash signature scheme. See Section 4.4.2). This is achieved in our constructions.

Finally, we analyze and state all our security results exactly (as opposed to asymptotically) and strive for the best possible reductions.[2]

1.6 An interesting open question

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 I am sending the same message to many parties except with different headers, I sign one copy and update to obtain the rest. But I keep the original copy and its signature on my machine— when I update I know the original is authentic.

But there are some situations in which one might want an even stronger form of security. For example, suppose you are remote editing a file residing on an insecure machine, and at any time the machine could be hit by a virus which would tamper with the data. For efficiency you are incrementally signing the file every time you make a change to it. But when you run the update algorithm, you 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).

We formalize a new notion of security under *substitution attacks* appropriate to the above setting. We then show that substitution attacks can be used to break the above hash-and-sign scheme when the hash function is our 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!

We leave it as an open problem to find ideal incremental signature schemes secure against substitution attack.

[2] Exact security is not new. Although the majority of theoretical works only make asymptotic statements, the exact tradeoffs can be derived from the proofs. (However these tradeoffs are sometimes quite bad). Moreover several previous works explicitly address exact security with concern for tradeoff quality, eg. [12, 14, 17, 11, 1].

2 Preliminaries

We follow the notation for algorithms and probabilistic experiments that originates in [13] and refer the reader there for a detailed exposition. Let's briefly recall that $z \stackrel{R}{\leftarrow} A(x, y, \cdots)$ is the experiment of running probabilistic algorithm A and letting z be its output, and $[A(x, y, \cdots)]$ is the set of all strings output by $A(x, y, \cdots)$ with positive probability. PPT denotes "probabilistic, polynomial time."

Our results require us to be careful about the model of computation. Rather than the traditional Turing machine model, we use the (in any case more realistic) RAM model. In this model any algorithm $A(x, y, \cdots)$ has random access to each of of its inputs $x, y, \cdots$.

A message is viewed as a sequence of blocks. The block size is denoted b and we let $B_b = \{0,1\}^b$ be the domain within which blocks range. We let n denote the number of blocks and B_b^n the space of n-block messages. With b understood, $M[i]$ is the i-th block of $M \in B_b^n$.

A *replacement request* has the form (j, m) with $1 \leq j \leq n$ and $m \in B_b$. We let $M\langle j, m\rangle$ denote the message consisting of M with block j replaced by m. We'll say M has been *updated* or *incremented* by (j, m).

3 Incremental collision-free hashing

3.1 Families of hash functions

We need to extend usual definitions of hash families 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 3.1 A family of hash functions is specified by a pair $\mathcal{H} = (\mathsf{HGen}, \mathsf{HEval})$ 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 $M \in B_b^n$ and outputs a k bit string called the hash of M under H.

When the family $(\mathsf{HGen}, \mathsf{HEval})$ is clear from the context, we will identify H with $\mathsf{HEval}(H, \cdot)$ and regard it as a map of B_b^n to $\{0,1\}^k$. In particular we will write $H(M)$ for $\mathsf{HEval}(H, M)$.

3.2 Incrementality

The following definition says that an update algorithm IncH is one that can turn the hash of M into the hash of $M\langle j, m\rangle$.

Definition 3.2 Let $\mathcal{H} = (\mathsf{HGen}, \mathsf{HEval})$ specify a family of hash functions. We say that IncH is an update algorithm for $\mathcal{H}$ with running time $T(\cdot, \cdot, \cdot)$ if

$$\forall\, k, b, n, \;\; \forall\, H \in [\mathsf{HGen}(1^k, 1^b, 1^n)] \;\; \forall\, j \in \{1, \ldots, n\}, \;\; \forall\, m \in B_b,$$

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

The IncH-*augmentation* of $\mathcal{H} = (\mathsf{HGen}, \mathsf{HEval})$ is the triple $\mathcal{H}^+ = (\mathsf{HGen}, \mathsf{HEval}, \mathsf{IncH})$.

Notice this definition makes no requirement on the running time T of IncH. So, in particular, an update algorithm can just run $\mathsf{HEval}(H, M\langle j, m\rangle)$ to compute its output. We don't wish to exclude this— it is a legitimate update algorithm. But of course an update algorithm will be interesting only when it runs faster than HEval. The term "incremental hash family" will be loosely used to refer to a hash family possessing some "non-trivial" update algorithm.

We would like to say 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 we need the RAM model as opposed to the Turing machine model) and do some quick computation before writing the output back to memory. The formal definition that follows, however, allows a dependence of the time on $\log n$ (because otherwise the blocks cannot be accessed even in a RAM) but this quantity will be much smaller than, say, $k + b$, in any realistic situation, and thus we view the running time as being independent of n.

Definition 3.3 An update algorithm IncH for $\mathcal{H}$ is ideal if its running time $T(k, b, n)$ is polynomial in k, b and $\log n$.

We'll say that $\mathcal{H}$ is an *ideal incremental* scheme if it possesses an ideal update algorithm.

3.3 Collision-freeness

Incrementality does not necessitate any additions to the usual notions of attacks on the hash family. (This is in contrast to the situation for signatures, where the presence of incrementality will introduce new security issues). Thus we just have the usual notion of collision-freeness. A *collision-finder* for $\mathcal{H}^+$ is a probabilistic algorithm A. We discuss security via the following experiment.

Experiment describing A's attack on $\mathcal{H}^+(k, b, n)$:–

(1) Run $\mathsf{HGen}(1^k, 1^b, 1^n)$ to get H.

(2) Run A on input H. We ask that at the end of her execution, A outputs a pair of *distinct* messages $M_1, M_2 \in B_b^n$. We say A is *successful* if $H(M_1) = H(M_2)$.

We say that A *succeeds in* (t, ϵ)*-breaking* $\mathcal{H}^+(k, b, n)$ if, in the above experiment, A runs for time t and is successful with probability at least ϵ. We don't say what it means for a scheme to be "secure:" there is no need, because we will make stronger statements on the exact security (cf. Theorem 3.4) which imply the usual asymptotic notion.

The fact that the hash function is "public" is captured in the fact that the adversary is given its descripton when trying to find collisions.

3.4 An incremental hash family

3.4.1 *Discrete log in groups of prime order*

We fix a PPT algorithm $\mathsf{PrimeGen}$ which on input 1^k outputs a $k+1$ bit prime p identifying a group G_p of (prime) order p. We let $\mathcal{G} = \bigcup_k \mathcal{G}(k)$, where $\mathcal{G}(k) = \{\, G_p : p \in [\mathsf{PrimeGen}(1^k)]\,\}$, be the set of all these groups, and we assume the discrete log problem for $\mathcal{G}$ is hard (when the prime is chosen according to PrimeGen). Such groups have been used for cryptography by Croft and Harris [8], Schnorr [17], Chaum and Van Antwerpen [6], and others, and we refer the reader to these works for how to choose such groups. In particular, with appropriate assumptions on the distribution of primes if necessary, it can be done so that G_p is a subgroup of Z_q^* for some q of size $O(k)$, so that we may assume efficient group operations. In particular exponentiation takes $O(k^3)$ time.

A key fact is that since G_p has prime order, every non-trivial element is a generator. We let $\mathsf{index}_g^{G_p}(x) \in \{0, 1, \ldots, p-1\}$ denote the discrete logarithm of x to (non-trivial) base g in the group G_p. A *discrete log finder* is a probabilistic algorithm B.

Experiment describing B's attack on $\mathcal{G}(k)$:–

(1) Let $p \stackrel{R}{\leftarrow} \mathsf{PrimeGen}(1^k)$; $g \stackrel{R}{\leftarrow} G_p - \{1\}$; $x \stackrel{R}{\leftarrow} G_p - \{1\}$.

(2) Run B on input p, g, x. We say that she is successful if her output is $\mathsf{index}_g^{G_p}(x)$.

We way that B *succeeds in* (t, ϵ)*-breaking* $\mathcal{G}(k)$ if in the above experiment she halts in t steps and is successful with probability at least ϵ.

We denote by $\langle\cdot\rangle$: $B_b \to \{1, \ldots, 2^b\}$ an encoding of message blocks into non-zero integers. To be specific, we set $\langle m \rangle$ to 1 plus the number whose binary expansion is m. Thus for any prime p of length at least $b+1$ and any $g \in G_p$ we can compute $g^{\langle m \rangle}$, and if g is non-trivial so is $g^{\langle m \rangle}$.

3.4.2 *The hash family and the update algorithm*

The block size will be set equal to the security parameter, $b = k$. (Formally have the following algorithms output junk when $b \neq k$. Theorem 3.4 only addresses the case $b = k$).

The hash family $\mathcal{H} = (\mathsf{HGen}, \mathsf{HEval})$ is specified as follows. On input $1^k, 1^k, 1^n$ the generator HGen runs $\mathsf{PrimeGen}(1^k)$ to get a $k+1$ bit prime p. It then selects $g_1, \ldots, g_n$ at random from $G_p - \{1\}$ and outputs $(p\,;\, g_1, \ldots, g_n)$ as the description of the hash function H. The value of the hash function $H = (p\,;\, g_1 \ldots g_n)$ on a given message $M = M[1] \ldots M[n] \in B_k^n$ is given by

$$H(M[1] \ldots M[n]) \stackrel{\text{def}}{=} \mathsf{HEval}(H, M[1] \ldots M[n]) \;=\; \prod_{i=1}^{n} g_i^{\langle M[i] \rangle} \;,$$

the operations being of course in the group G_p.

The interest of [7] in this family seemed to stem from its efficiency as compared, for example, to that of the (discrete log based) hash family of [9]. Brands [5] mentions the family in the context of a general exposition of the "representation" problem. The (seemingly rare) incremental property that we next observe it possesses seems for the first time to pinpoint a crucial use of this family, and in some sense answers a question of [5] who asked for interesting uses of the representation problem when n was more than a constant.

Define the algorithm IncH by

$$\mathsf{IncH}(H, M, h, (j, m)) \;=\; h \cdot g_j^{-\langle M[j] \rangle} \cdot g_j^{\langle m \rangle} \;.$$

It is easy to see that if $h = H(M)$ then $\mathsf{IncH}(H, M, h, (j, m)) = H(M\langle j, m \rangle)$. Now note IncH can random access the $O(nk)$ bit description of H to get g_j in $\mathrm{poly}(k, \log n)$ time, and similarly for the other inputs. Its output is then given by a polynomial time computation on $O(k)$ bit inputs and hence the algorithm runs in $\mathrm{poly}(k)$ time. Thus it is an ideal update algorithm for $\mathcal{H}$.

3.4.3 *Security*

A proof of the security of $\mathcal{H}^+$ takes the following form. Given a collision finder A for $\mathcal{H}^+(k,k,n)$ we construct a discrete log finder B for $\mathcal{G}(k)$. Now suppose A succeeds in (t,ϵ)-breaking $\mathcal{H}^+(k,k,n)$. The question we consider is for what values of t',ϵ' the constructed algorithm B succeeds in (t',ϵ')-breaking $\mathcal{G}(k)$.

Previous works [7, 5] have only discussed asymptotic security, where one sets $n = n(k)$ to some fixed polynomial in k, regards t,ϵ,t',ϵ' 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 we may know what size to choose for the prime p and what adversaries we can tolerate with a specific security parameter. Moreover, it is important to strive for the tightest possible reduction, because this means that the same "security" can be obtained with a smaller value of the security parameter, meaning greater efficiency. Thus we want 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 [7] 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 [5] proposes a reduction which removes the restriction on n and achieves $t' = t + O(nk^3)$ and $\epsilon' = \epsilon/n$. The running time of B here is essentially optimal: we 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 our case) n may be very large. Thus even if A is successful with high probability, the above may only let us conclude that B is successful with low probability.

We improve the reduction to be essentially optimal. We preserve the current quality of the running time, and 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. U denotes some oracle machine which depends only on our proof and the given family $\mathcal{H}$. 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. $\mathcal{H}^+$ is the IncH-augmentation of $\mathcal{H}$.

Theorem 3.4 There is an oracle machine U such that the following is true. Suppose collision-finder A succeeds in (t,ϵ)-breaking $\mathcal{H}^+(k,k,n)$. Then discrete log finder $B \stackrel{\text{def}}{=} U^A$ succeeds in (t',ϵ')-breaking $\mathcal{G}(k)$ where $t' = t + O(nk^3)$ and $\epsilon' = \epsilon/2$.

The proof is in Appendix A.

3.4.4 *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 assume we are in a setting with frequent updates. With MD5 we have no choice but to hash from scratch, while in our scheme we can use the update algorithm to update the hash in two exponentiations. Thus to compare the efficiency we 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 $512n$ bit string. A computation yields a reasonable value.

Note 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 [4].

3.4.5 *A practical version with small description size*

The size of (the description of) the hash function in the above is $O(nk)$ so that it depends on the message size, which we assume large. In practice this is too much. Here we suggest a way to reduce the size to $O(k)$. We let f: $\{0,1\}^k \to \{0,1\}^{O(k)}$ 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 $g_1, \ldots, g_n$ for themselves. 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 our proof of security to this setting is not difficult and we can conclude (the following statement is informal) that the scheme just described satisfies Theorem 3.4 in the random oracle model. As discussed by [2], 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. We refer the reader to this paper also for more suggestions on functions with which to "instantiate" f.

4 Incremental Signing

4.1 Signature schemes

Definition 4.1 A signature scheme is a triple $\mathcal{S} = (\mathsf{KGen}, \mathsf{Sig}, \mathsf{Vf})$ of algorithms. There is a polynomial $s(\cdot,\cdot,\cdot)$ called the **signature size** such that

- The PPT **key generator** KGen takes as input $1^k, 1^b, 1^n$ and outputs a pair (Sk, Vk) of strings called, respectively, the **signing** and (corresponding) **verifying** keys.
- The PPT **signing** algorithm Sig takes as input Sk and $M \in B_b^n$ and outputs a $s(k,b,n)$-bit string called the **signature** of M.
- The polynomial time **verifying** algorithm Vf outputs a bit and satisfies $\mathsf{Vf}(Vk, M, \sigma) = 1$ for every $M \in B_b^n$ and every $\sigma \in [\mathsf{Sig}(Sk, M)]$.

The assumption that Vf is deterministic is for simplicity only: in general one can consider probabilistic verifiers.

We'll say that a signature scheme has *short signatures* if the signature size depends only on k. In such a case we abuse notation and write the signature size as $s(k)$.

4.2 Incrementality

An update algorithm is one that can turn a signature of M into *some* signature of $M\langle j, m\rangle$.

Definition 4.2 Let $\mathcal{S} = (\mathsf{KGen}, \mathsf{Sig}, \mathsf{Vf})$ be a signature scheme. We say that IncSig is an update $\mathsf{algorithm}$ for $\mathcal{S}$ with running time $T(\cdot, \cdot, \cdot)$ if

$$\forall\, k, b, n, \ \ \forall\, (Sk, Vk) \in [\mathsf{KGen}(1^k, 1^b, 1^n)] \ \ \forall\, j \in \{1, \ldots, n\}, \ \ \forall\, m \in B_b,$$

if $\mathsf{Vf}(Vk, M, \sigma) = 1$ then it is the case that $\mathsf{IncSig}(Sk, M, \sigma, (j, m))$ halts in $T(k, b, n)$ steps with output σ' satisfying $\mathsf{Vf}(Vk, M\langle j, m\rangle, \sigma') = 1$.

The IncSig-*augmentation* of $\mathcal{S} = (\mathsf{KGen}, \mathsf{Sig}, \mathsf{Vf})$ is $\mathcal{S}^+ = (\mathsf{KGen}, \mathsf{Sig}, \mathsf{Vf}, \mathsf{IncSig})$.

Note that the output of $\mathsf{IncSig}(Sk, M, \sigma, (j, m))$ is not required to be distributed in the same way as that of $\mathsf{Sig}(Sk, M\langle j, m\rangle)$— IncSig just has to return something that Vf would accept.

The term "incremental signature scheme" will be loosely used to refer to a signature scheme possessing some "non-trivial" update algorithm. Ideality of an update algorithm is defined in analogy to Definition 3.3, and an ideal incremental signature scheme is one that possesses an ideal update algorithm.

The schemes we prefer have short signatures, but it is possible to discuss update algorithms (even ideal ones) even if the signatures are long. In such a case IncSig will not be able to output the entire signature— one imagines that it modifies σ in a few chosen places and the result is what we view as σ'.

Analogously one can define the notion of incremental verification. We leave it to the reader.

4.3 Basic security

We recall that we will be evaluating the security of signature schemes at two levels, motivated by differing security demands of applications. The basic level we present here is suitable for settings in which a signer updating signature σ of message M is guaranteed that these quantities are authentic. In the majority of applications of digital signatures this assumption is valid.

The definition extends the notion of existential forgery under adaptive chosen message attack to allow the adversary access to $\mathsf{IncSig}(Sk, \cdots)$. (This is necessary because signatures produced by IncSig might be from a different distribution than those produced by Sig and perhaps the adversary can gain an advantage by seeing examples from this new distribution). The restriction that updates only be made on authentic data is captured below in the fact that the incremental signing requests simply point to a message and signature from the past.

Experiment describing F's attack on $\mathcal{S}^+(k, b, n)$:–

(1) Run $\mathsf{KGen}(1^k, 1^b, 1^n)$ to get keys Sk, Vk.

(2) Initialize: Set $\alpha = 0$.

(3) Run the adversary F on input Vk. Her oracle queries are answered as follows.

 (3.1) Suppose F makes a simple signing request— this has the form of a message $M \in B_b^n$. Let $\sigma \overset{R}{\leftarrow} \mathsf{Sig}(Sk, M)$ and return σ to F. Let $\alpha \leftarrow \alpha + 1$. Let $M_\alpha \leftarrow M$ and $\sigma_\alpha \leftarrow \sigma$.

(3.2) Suppose F makes an incremental signing request— this has the form $((j,m),\beta)$ with $\beta \in \{1,\ldots,\alpha\}$. Let $\sigma' \stackrel{R}{\leftarrow} \mathsf{IncSig}(Sk, M_\beta, \sigma_\beta, (j,m))$ and return σ' to F. Let $\alpha \leftarrow \alpha + 1$. Let $M_\alpha \leftarrow M_\beta\langle j,m\rangle$ and $\sigma_\alpha \leftarrow \sigma'$.

(4) We ask that at the end of her execution, F output a pair (M,σ) such that $M \notin$ Legal, where Legal $= \{ M_1,\ldots,M_\alpha \}$. We say that F is successful if $\mathsf{Vf}(Vk, M, \sigma) = 1$.

We say that F *succeeds in* $(t, q_{\rm sig}, q_{\rm inc}, \epsilon)$*-breaking* $\mathcal{S}^+(k,b,n)$ *with a basic attack* if, in the above experiment, she runs for t steps, makes $q_{q_{\rm sig}}$ simple signing requests, makes $q_{q_{\rm inc}}$ incremental signing requests, and succeeds with probability at least ϵ.

4.4 Incremental signature schemes achieving basic security

In what follows $\mathcal{S}^* = (\mathsf{KGen}^*, \mathsf{Sig}^*, \mathsf{Vf}^*)$ denotes a standard (ie. not necessarily incremental) signature scheme as per Definition 4.1, assumed secure against existential forgery under adaptive chosen message attack in the standard sense of [13]. Exact security is discussed by saying that an adversary F^* *succeeds in* (t,q,ϵ)*-breaking* $\mathcal{S}^*(k,b,n)$ *with an adaptive chosen message attack* if in this attack she runs in time t, makes q signing queries, and succeeds in existential forgery with probability at least ϵ. We consider two standard transformations.

4.4.1 *Incremental hash-and-sign*

Given an incremental hash function, a slight variation of the standard hash-and-sign method yields an incremental signature scheme. Security must however be reconsidered, in light of the fact that our basic attacks allow attacks on the update algorithm. Luckily they do not cause any damage. For completeness we provide details below.

Let $\mathcal{H}^+ = (\mathsf{HGen}, \mathsf{HEval}, \mathsf{IncH})$ be a family of hash functions together with an update algorithm. We specify $\mathcal{S}^+ = (\mathsf{KGen}, \mathsf{Sig}, \mathsf{Vf}, \mathsf{IncSig})$ as follows.

On input $1^k, 1^b, 1^n$ algorithm KGen runs $H \stackrel{R}{\leftarrow} \mathsf{HGen}(1^k, 1^b, 1^n)$; $(Sk^*, Vk^*) \stackrel{R}{\leftarrow} \mathsf{KGen}^*(1^k, 1^k, 1^1)$. It outputs the signing key $Sk = (Sk^*, H)$ and the verifying key $Vk = (Vk^*, H)$. Note the keys of the original signature scheme are chosen to sign messages consisting of one k-bit block only.

The signature of $M \in B_b^n$ given the above keys is $\mathsf{Sig}(Sk, M) = (H(M), \mathsf{Sig}^*(Sk^*, H(M)))$. Namely the hash of the message, together with its signature under the original scheme. (Including the hash $h = H(M)$ in the signature is the slight variation. It may seem redundant since anyone can compute it given M, H, but it is important for incrementality). Note Sig^* is being applied only to a k-bit string.

Given the verification key $Vk = (Vk^*, H)$ and a string $\sigma = (h, \sigma^*)$ the algorithm Vf outputs 1 iff $h = H(M)$ and $\mathsf{Vf}^*(Vk^*, \sigma^*) = 1$.

Given $Sk, M, \sigma, (j,m)$ (with $Sk = (H, Sk^*)$ and $\sigma = (h, \sigma^*)$) the update algorithm IncSig first updates the hash by $h' = \mathsf{IncH}(H, M, h, (j,m))$. Then it computes from scratch a signature $\sigma' = \mathsf{Sig}^*(Sk^*, h')$ of (the k-bit string) h' under the original scheme. It outputs (h', σ').

Note that the signatures in this scheme are short, namely poly(k) bits.

The following theorem says that if $\mathcal{S}^+$ can be broken then either $\mathcal{S}^*$ or $\mathcal{H}^+$ can be broken, and specifies the exact security corresponding to this statement. The function $\tau(k,b,n)$ represents time depending only on the algorithms defining the schemes. It should be viewed as much smaller than

t and its exact value can be derived from the proof.

Theorem 4.3 There are oracle machines U_1, U_2 and a function $\tau(k, b, n)$ such that the following is true. Suppose F succeeds in $(t, q_{\rm sig}, q_{\rm inc}, \epsilon)$-breaking $\mathcal{S}^+(k, b, n)$ with a basic attack, and let $q = q_{\rm sig} + q_{\rm inc}$. Then one of the following is true:

(1) Either $F^* \stackrel{\rm def}{=} U_1^F$ succeeds in $(t + q \cdot \tau(k, b, n), q, \epsilon/2)$-breaking $\mathcal{S}^*(k, k, 1)$ with an adaptive chosen-message attack, or

(2) $A \stackrel{\rm def}{=} U_2^F$ succeeds in $(t + q \cdot \tau(k, b, n), \epsilon/2)$-breaking $\mathcal{H}^+(k, b, n)$.

The proof of Theorem 4.3 is in Appendix B.

The hardness of discrete log implies, via [15], the existence of standard (ie. non-incremental) signature schemes which can play the role of $\mathcal{S}^*$ in the above. Combining this with the results of Section 3.4.3 we have 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 [15]. For a practical version we could use El Gamal's scheme [10] or RSA in the role of $\mathcal{S}^*$ and the practical version of our hash function (cf. Section 3.4.5) in the role of $\mathcal{H}$.

The public file is large because the hash function has poly(n, k) size. But it isn't necessary that each user publish 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).

4.4.2 *The tree hash scheme uses too much memory*

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 (ie. not necessarily incremental) collision-free hash function $H\colon \{0,1\}^{2k} \to \{0,1\}^k$. 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 (cf. Section 4.3) but some thought shows that the scheme satisfies our 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 our formal definition, it is too memory inefficient to be interesting in most applications. We want schemes with short signatures and hence prefer the method of Section 4.4.1.

4.5 Security against substitution attacks

We provide here a stronger notion of security for incremental signature schemes, suitable for applications like remote editing a file on an insecure machine. We let $\mathcal{S}^+ = (\mathsf{KGen}, \mathsf{Sig}, \mathsf{Vf}, \mathsf{IncSig})$ be an augmented signature scheme. The adversary's incremental signing requests now have a new form: she supplies $M, \sigma, (j, m), \beta$. We first describe the experiment then provide explanation and discussion.

Experiment describing F's attack on $\mathcal{S}^+(k, b, n)$:–

(1) Run $\mathsf{KGen}(1^k, 1^b, 1^n)$ to get keys Sk, Vk.

(2) Initialize: Set $\alpha = 0$.

(3) Run the adversary F on input Vk. Her oracle queries are answered as follows.

- (3.1) Suppose F makes a simple signing request— this has the form of a message $M \in B_b^n$. Let $\sigma \stackrel{R}{\leftarrow} \mathsf{Sig}(Sk, M)$ and return σ to F. Let $\alpha \leftarrow \alpha + 1$ and let $M_\alpha \leftarrow M$.
- (3.2) Suppose F makes an incremental signing request— this has the form $(M, \sigma, (j, m), \beta)$ with $\beta \in \{1, \ldots, \alpha\}$. Let $\sigma' \stackrel{R}{\leftarrow} \mathsf{IncSig}(Sk, M, \sigma, (j, m))$ and return σ' to F. Let $\alpha \leftarrow \alpha + 1$ and let $M_\alpha \leftarrow M_\beta\langle j, m \rangle$.

(4) We ask that at the end of her execution, F output a pair (M, σ) such that $M \notin \text{Legal}$, where $\text{Legal} = \{ M_1, \ldots, M_\alpha \}$. We say that F is successful if $\mathsf{Vf}(Vk, M, \sigma) = 1$.

We say that F *succeeds in* $(t, q_{\text{sig}}, q_{\text{inc}}, \epsilon)$*-breaking* $\mathcal{S}^+(k, b, n)$ *with a substitution attack* if, in the above experiment, she runs for t steps, makes $q_{q_{\text{sig}}}$ simple signing requests, makes $q_{q_{\text{inc}}}$ incremental signing requests, and succeeds with probability at least ϵ.

Recall that the assumption in basic security was that when the signer applies the update algorithm it is to "authentic" data. We are assuming we are in a situation where this assumption is not realistic; for example, the data is on an insecure medium and when the signer accesses it to update a message and signature, he cannot be sure it has not been tampered with. In the worst case, he must assume it has been adversarially tampered with.

To model this the adversary is asked, as before, to point, via β, to that message out of the past on which she is requesting an update, and to supply the update request (j, m). The novel element is that she will additionally supply M, σ, to be taken to mean that she has substituted these for M_β, σ_β. That is, she has tampered with the data.

The index β is not reflected in the way her query is answered— the answer is obtained by applying $\mathsf{IncSig}(Sk, \cdots)$ to the message M and accompanying $\sigma, (j, m)$ that F provides. But β is used to update the signer's own "view" of what is happening. The idea is that that signer has "accepted" to update M_β according to (j, m), and thus has, from his point of view, willingly signed $M_\beta\langle j, m \rangle$. In other words, we can view the set Legal, at the end of the experiment, as being all those messages which the signer believes he has signed.

The notion of existential forgery says F is successful if she outputs a message M not previously queried of $\mathsf{Sig}(Sk, \cdot)$, and passing verification. We recall that the intuition is that "legitimately signed" messages are excluded. Thus according to the above discussion, we should declare F successful if she forges the signature of a message not in Legal.

Why would such an attack help the adversary? The reason is that IncSig was designed to be used on inputs $Sk, M, \sigma, (j, m)$ for which $\mathsf{Vf}(Vk, M, \sigma) = 1$, and we don't know what happens when this algorithm is run on strange inputs. One might ask why IncSig doesn't simply check that $\mathsf{Vf}(Vk, M, \sigma) = 1$. The reason is that in general this could defeat the efficiency we are trying to gain. For example, if IncSig is ideal it has only $\mathrm{poly}(k, b, \log n)$ time and verification takes $\mathrm{poly}(k, b, n)$ time.

It is important to note that we do not view the adversary as having legitimately obtained the signature of $M\langle j, m\rangle$— what the signer believes he has signed is $M_\beta\langle j, m\rangle$.

4.6 A successful substitution attack

We illustrate the strength of substitution attacks by showing how the scheme of Section 4.4.1 can be broken, in this setting, when we use, as the hash family, the one of Section 3.4. (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.

For simplicity assume 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 $H(M) = g^{\langle M\rangle} = g^{1+M}$, the operations being in G_p. We let Sk^* be the signing key under the standard scheme, so that the signature of M is $\sigma = (g^{1+M}, \sigma^*)$ where $\sigma^* \stackrel{R}{\leftarrow} \mathsf{Sig}^*(Sk^*, g^{1+M})$.

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. We set $M_1 = A$.

Now, F make 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 our scheme, 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 $\sigma_F^* \stackrel{R}{\leftarrow} \mathsf{Sig}^*(Sk^*, h_F)$. 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. That is, in terms of the experiment of Section 4.5, we have $M_2 = C$. 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 $\mathrm{Legal} = \{A, C\}$. For appropriate choices of B, C (it suffices that $B \notin \{A, C\}$) it is the case that $A - B + C \notin \mathrm{Legal}$. 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^*.

We don't know 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 against substitution attack.

We leave as an open problem to design an incremental signature scheme secure against substitution attack, under the restrictions that the signature be short and the update algorithm be ideal. Some progress towards this question is described below.

5 Work in progress

In [3] we expand the scope of this research in the following directions. First, we consider more complex update operations on messages such as insertion (of a new block into the message) or deletion (of an existing block). These are clearly important in applications. Second, we consider other primitives such as finger-printing and message authentication. We appropriately extend the notion of subsitution attack to these contexts. Our main result is a finger-printing scheme which permits insertion and deletion and is secure against substitution attack.

Acknowledgments

We thank Hugo Krawczyk for many informative discussions on this materiel.

References

[1] M. Bellare, J. Kilian and P. Rogaway. The security of cipher block chaining. *Advances in Cryptology – Crypto 94 Proceedings*, Lecture Notes in Computer Science Vol. 839, Springer-Verlag, Y. Desmedt, ed., 1994.

[2] 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.

[3] M. Bellare, O. Goldreich and S. Goldwasser. Work in progress.

[4] 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.

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

[6] 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.

[7] 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.

[8] Croft and Harris. Public key cryptography and re-usable shared secrets. In *Cryptography and Coding*, Clarendon Press, 1989.

[9] I. Damgård. 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.

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

[11] S. Even, O. Goldreich and S. Micali. On-line/Off line digital signatures. Manuscript. Preliminary version in Crypto 89.

[12] 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.

[13] 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.

[14] R. Impagliazzo, L. Levin and M. Luby. Pseudo-random generation from one-way functions. *Proceedings of the Twenty First Annual Symposium on the Theory of Computing,* ACM, 1989.

[15] 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.

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

[17] 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.

A Proof of Theorem 3.4

We first describe the algorithm $B = U^A$. Then we argue that its running time is as claimed and finally that its success probability is as claimed.

On inputs p, g, x algorithm B selects $r_1, \ldots, r_n \in \{0,1\}$ at random and $u_1, \ldots, u_n \in \{0, 1 \ldots, p-1\}$ at random. For $i = 1, \ldots, n$ it sets

$$g_i \;=\; \begin{cases} g^{u_i} & \text{if } r_i = 0 \\ x^{u_i} & \text{if } r_i = 1\,. \end{cases}$$

It sets $H = (p\,;\, g_1, \ldots, g_n)$. Now it invokes $A(H)$ and obtains distinct messages

$$M_1 \;=\; M_1[1] \ldots M_1[n] \quad \text{and} \quad M_2 \;=\; M_2[1] \ldots M_2[n]\,. \tag{1}$$

For $j = 1, 2$ it is now convenient to set $t_{j,i} = \langle M_j[i] \rangle$. Algorithm B sets $a = \sum_{r_i=1} u_i(t_{1,i} - t_{2,i})$, 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. (That is, $ba \equiv 1 \bmod p$. Such an inverse always exists since p is prime, and it can be found via Euclid's algorithm). B outputs $\alpha = b \cdot \sum_{r_i=0} u_i(t_{2,i} - t_{1,i}) \bmod p$ 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 —ie. $H(M_1) = H(M_2)$— 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 re-arranging 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. We now claim that with probability at least 1/2 we have $a \neq 0$. Given this, raise both sides of the above equation to the power b to get

$$x = x^{ab} = \prod_{r_i=0} g^{bu_i(t_{2,i}-t_{1,i})} = g^{\alpha} ,$$

showing that α is indeed $\mathsf{index}_g^{G_p}(x)$. It remains to justify the claim. We will argue this informally. We will use the following technical fact.

Technical Fact. Let $a_1, \ldots, a_n$ be numbers with the property that $\sum_{i=1}^{n} a_i \neq 0$. 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 $X = \sum_{i=1}^{n} X_i$. Then $\Pr[\, X \neq 0 \,] \geq 1/2$.

We note that the distribution on $g_1, \ldots, g_n$ is independent of $r_1, \ldots, r_n$. Thus we may think of the experiment as the following game. We choose $g_1, \ldots, g_n$ at random and obtain the collision from A. We let $a_i = u_i(t_{1,i} - t_{2,i})$ for $i = 1, \ldots, n$. Now we choose $r_1, \ldots, r_n$ at random and compute $\sum_{r_i=1} a_i$. Viewed this way we can see it is the same as the technical fact stated above.

B Proof of Theorem 4.3

Refer to Section 4.3 for the description of the experiment that describes F's attack on $\mathcal{S}^+(k, b, n)$. We leave the reader to check that the following algorithms meet the claim.

The algorithm F^*. On input a verification key Vk^* of the original scheme, and with oracle access to $\mathsf{Sig}^*(Sk^*, \cdot)$, algorithm F^* runs $\mathsf{HGen}(1^k, 1^b, 1^n)$ to get H. She sets $Vk = (Vk^*, H)$ and runs F on this input. Set $\alpha = 0$. F's oracle queries are dealt with as follows:

- Suppose F makes the simple signing request M. F^* lets $\alpha \leftarrow \alpha + 1$ and $M_\alpha \leftarrow M$. Then F^* computes $h_\alpha = H(M)$ and invokes its oracle to obtain $\sigma_\alpha^* = \mathsf{Sig}^*(Sk^*, h_\alpha)$. She returns $\sigma_\alpha = (h_\alpha, \sigma_\alpha^*)$ to F.
- Suppose F make the incremental signing request $((j, m), \beta)$. F^* lets $\alpha \leftarrow \alpha + 1$ and $M_\alpha \leftarrow M_\beta\langle j, m\rangle$. Then F^* computes $h_\alpha = \mathsf{IncH}(H, M_\beta, h_\beta, (j, m))$ and invokes its oracle to obtain $\sigma_\alpha^* = \mathsf{Sig}^*(Sk^*, h_\alpha)$. She returns $\sigma_\alpha = (h_\alpha, \sigma_\alpha^*)$ to F.

Suppose F eventually halts with output (M, σ) where $\sigma = (h, \sigma^*)$. Then F^* outputs (h, σ^*) as his forgery, and halts.

The algorithm A. On input H chosen according to $H \stackrel{R}{\leftarrow} \mathsf{HGen}(1^k, 1^b, 1^n)$, algorithm A lets $(Sk^*, Vk^*) \stackrel{R}{\leftarrow} \mathsf{KGen}^*(1^k, 1^k, 1^1)$. She sets $Vk = (Vk^*, H)$ and runs F on this input. F's oracle queries which are dealt with as follows:

- Suppose F makes the simple signing request M. A lets $\alpha \leftarrow \alpha + 1$ and $M_\alpha \leftarrow M$. Then A computes $h_\alpha = H(M)$ and $\sigma_\alpha^* = \mathsf{Sig}^*(Sk^*, h_\alpha)$. She returns $\sigma_\alpha = (h_\alpha, \sigma_\alpha^*)$ to F.

- Suppose F makes the incremental signing request $((j, m), \beta)$. A lets $\alpha \leftarrow \alpha + 1$ and $M_\alpha \leftarrow M_\beta\langle j, m\rangle$. Then A computes $h_\alpha = \mathsf{IncH}(H, M_\beta, h_\beta, (j, m))$ and $\sigma^*_\alpha = \mathsf{Sig}^*(Sk^*, h_\alpha)$. She returns $\sigma_\alpha = (h_\alpha, \sigma^*_\alpha)$ to F.

Suppose F eventually halts with output (M, σ) where $\sigma = (h, \sigma^*)$. If $h \notin \{h_1, \ldots, h_\alpha\}$ then A halts with arbitrary output; else she chooses i such that $h = h_i$, outputs (M_i, M), and halts.