available at www.sciencedirect.com
journal homepage: www.elsevier.com/locate/cose

Computers & Security

Security considerations for incremental hash functions based on pair block chaining

Raphael C.-W. Phan[a,*], David Wagner[b]

[a]Information Security Research (iSECURES) Lab, Swinburne University of Technology (Sarawak Campus), 93576 Kuching, Malaysia
[b]Computer Science Division, University of California, Berkeley, CA 94720, USA

ARTICLE INFO

Article history:
Received 24 August 2004
Revised 10 November 2005
Accepted 2 December 2005

Keywords:
Cryptography
Hash function
Incremental cryptography
Collision resistance
Pair block chaining

ABSTRACT

Incremental hash functions have gained much attention due to their incremental property, i.e. hashes of updated messages can be speedily computed from previous hashes without having to re-hash the message as was the case in conventional hash functions. In this paper, we first show how collisions can be obtained in such incremental hash functions that are *based on pair block chaining*, highlighting that more caution should be taken into its design process. We then identify some design and implementation criteria for such incremental hash functions.

1. Introduction

Incremental hash functions (Bellare et al., 1994) were introduced in 1994, and are very much suited for situations where if a previously hashed message, M is slightly updated into a new message, M^*, then it should be fairly quick to compute the hash value of the updated message, M^*. This is done by computing the new hash, μ^*, from the old hash value, μ, in contrast to conventional hash functions that have to recompute the new hash, μ^* from scratch, which takes a longer time.

Applications of incremental hash functions include:

(a) Virus protection (Bellare et al., 1994, 1995a): viruses typically modify the host files that they infect, and so one way of virus detection involves checking files for signs of unauthorized modification. This can be done by using hash functions to compute authentication tags from each file. Unauthorized modifications to these tagged files would trigger an alarm since the authentication tags would no longer be valid. That being said, files within a computer system are often updated by the computer user and so it is desirable that their corresponding authentication tags be easily recomputed to reflect the updates.
Doing this with conventional hash functions is time-consuming since one would have to recompute the hash of the updated value from scratch, even though only a minor update was done on the authenticated file. Incremental hash functions solve this problem by easily recomputing the new hash directly from the old hash value. This is especially useful when using a processor with limited processing power and storage capacity, as is the case for smart cards. In such a case, one would not want to have to recompute the authentication tags attached to each file every time a file is updated.

* *Corresponding author*. Tel.: +60 82 416 353; fax: +60 82 426 353.
E-mail addresses: rphan@swinburne.edu.my (R.C.-W. Phan), daw@cs.berkeley.edu (D. Wagner).

0167-4048/$ – see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cose.2005.12.006

(b) Memory checkers (Fischlin, 1997): in a similar setting, one also desires to verify the integrity of memory locations used to store sensitive information such as financial data and passwords in banks, etc. Therefore, there is a need to be able to quickly check the integrity of memory for any unauthorized modifications.
(c) Broadcast networks (Bellare et al., 1994): in a situation where a similar message needs to be transmitted to different users, an incremental hash function would allow one to quickly recompute the hash of subsequent similar messages when the first message has already been hashed.
(d) Video surveillance broadcasting (Bellare et al., 1994): since video surveillance cameras typically consist of successive frames that differ only slightly from each other, and that need to be integrity-protected, the hashes of these frames can be quickly computed from the hashes of the other similar frames.

Incremental hash functions are therefore very useful in the practical sense to ensure fast integrity checks.

Some recent designs of incremental hash functions have been based on the concept of *pair block chaining* (Bellare et al., 1995a), e.g. the incremental XOR scheme (Bellare et al., 1995a) and PCIHF variants (Goi et al., 2001, 2003b), which involves taking two subsequent blocks (a pair) of a message, M, at a time, and feeding them into a pseudo-random function, R, before chaining all the outputs of R into the final hash, μ.

The purpose of this paper is to show that more caution should be taken when designing such incremental hash functions in order that they be collision-resistant. We demonstrate this fact by first showing three cases of how collisions can be obtained in incremental hash functions based on pair block chaining. Based on this, we highlight security considerations for constructing incremental hash functions, especially those based on pair block chaining. When necessary, we will cite examples of ways that specific schemes (Bellare et al., 1995a; Goi et al., 2001, 2003b) fail in order to illustrate and justify our points.

In Section 2, we briefly review incremental hash functions, the common paradigm used to construct them, the pair block chaining construction and specific schemes based on this. In Section 3, we discuss three cases that collisions will occur in incremental hash functions based on pair block chaining. We then give in Section 4 a summarized discussion of the security considerations for hash functions, starting with general ones, and then move onto those specific for incremental hash functions. We conclude in Section 5.

2. Incremental hash functions based on pair block chaining

Incremental hash functions typically follow the *randomize-then-combine* paradigm (Bellare et al., 1997) where a message to be hashed is first divided into blocks and each block is then in turn put through a randomizing function, R, before the output blocks are all combined to form the final hash output, μ.

The *pair block chaining* (Bellare et al., 1995a; Goi et al., 2001) is a type of randomizing function on a sequence of n blocks of a message, M. The idea is to pair up two subsequent message blocks prior to inputting each pair to a pseudo-random function, R, and then combining all the outputs for a resultant hash value, μ. Supposing that a message, M can be broken up into n blocks, then the hash, μ, is computed as follows:

$$\mu = \bigoplus_{i=1}^{n-1} R(M[i] \| M[i+1]) \tag{1}$$

Note here that $\|$ denotes concatenation, and in this case, the combining function is an exclusive-OR (XOR) sum of all randomized blocks.

This idea of pair block chaining was first applied in the incremental XOR scheme (Bellare et al., 1995a) in 1995 for use to compute the signatures[1] (characteristics) of files to check for their integrity.

Some other combining operations besides XOR can be used, for example the PCIHF incremental hash function designed in 2001 (Goi et al., 2001) uses a similar pair block chaining construction, but is designed to be a fast and more secure alternative to the basic XOR scheme (Bellare et al., 1995a) by replacing the XOR combining sum with a modular addition sum.

Another PCIHF variant with a much larger modulus (e.g. moving from modulus 2^{160} to modulus 2^{1600}) was later proposed (Goi et al., 2003b) in 2003 to counter an attack by Wagner (2002) on the initially proposed version (Goi et al., 2001). Three methods were further given on how to implement this, which generally just combines every 10 consecutive 160-bit output blocks from the randomizing function, R (assuming SHA-1 is used as R) to form 1600-bit intermediate values for the final modulo 2^{1600} combining.

However, as we will demonstrate in the next section, more caution needs to be exercised when constructing incremental hash functions based on this pair block chaining technique, regardless of the combining function or the size of the modulus used.

3. Three cases of collisions

One major security criteria for hash functions is that they should be collision-resistant, which means that given two distinct messages, M and M', the probability that the corresponding hashes, μ and μ' are not equal is negligible. In this section, we demonstrate that for incremental hash functions *based on pair block chaining*, collisions would occur in three cases, namely messages with non-distinct blocks, when messages require padding and when cyclic chaining is used.

3.1. *Messages with non-distinct blocks*

We first consider the case of the incremental XOR scheme in Bellare et al. (1995a). For this, if some message blocks repeat (non-distinct), for instance, palindromic-like messages of the form $A \| B \| A$, where A, B represent any two message blocks, then there exists another message of the form $B \| A \| B$ such

[1] The term "signature" in this context does not refer to a digital signature, but corresponds more to the virus setting where it means the characteristic of a file, in this case, the hash of it.

that the two messages hash to the same value, μ, hence a collision occurs. Note that such collisions exist for palindromic messages of any block length that is not necessarily three as in the example given above. Also, non-palindromic messages equally fall to this attack, for example two messages of the form $A \| B \| C \| B \| A$ and $B \| C \| B \| A \| B$ also cause collisions. Basically, any two messages with the same block at both ends, and where all consecutively paired blocks follow the same order, would cause collisions. Finally, messages with repetitive blocks also fail. For example, two messages $A \| B \| B \| B \| B \| B \| C$ and $A \| B \| B \| B \| C$ would hash to the same value. This applies regardless of whether the repetitive blocks repeat an odd or even number of times, as long as they are either both odd or both even. The incremental XOR scheme (Bellare et al., 1995a) is therefore not collision-resistant.

Meanwhile, for PCIHF, this may also be a potential problem if not implemented carefully. Although the specification of PCIHF (Goi et al., 2001) rightly warns of certain inherent limitations such as this "strong assumption" that any two message blocks should be distinct, implementers and users are unlikely to see those warnings if they are not made explicit enough. Further, since this is a very strict and often impractical requirement, it should have been more clearly emphasized and highlighted as part of the implementation criteria. This raises the issue of the degree of explicitness that should be made by hash function designers in their proposals. Borrowing from the design criteria of authentication protocols (Lowe, 1996), we quote:

"...nothing in the description prohibits such an implementation. If the designer believes that certain implementation considerations are necessary, then these should be made explicit."

Explicitness in hash function design is clearly a practical concern. We have to bear in mind that most hash function implementers are not themselves the hash function designers, hence they would only follow whatever is stated in the hash function descriptions. Assumptions that hash function designers feel are obvious and trivial, might not be known by the implementers and hence would not be taken into consideration when these hash functions are implemented. In fact, this split in design and implementation is true for any area of security.

3.2. *Messages with padding*

When one designs incremental hash functions based on pair block chaining, checking that no two blocks are identical prevents collisions, as discussed previously in Section 3.1. Nevertheless, we show here that collisions would still occur when padding is required for messages that are not an integral length of the block size.

For instance, the PCIHF specification (Goi et al., 2001) suggests to pad such messages with a '1' followed by all '0's. For the sake of illustration, let the message block size, $b = 32$ bits. Let a message block $A = 1^2 0^{30}$ which denotes two '1's, followed by a string of 30 '0's. Let $B = 0^1 1^1 0^{30}$ which denotes a '0', a '1', followed by a string of 30 '0's. Then a 65-bit message of the form $A \| B \| 1$ would be padded to form $A \| B \| A$, while a 65-bit message of the form $B \| A \| 0$ would be padded to form $B \| A \| B$. This causes two such padded messages to hash to the same value, and a collision occurs. Note that this phenomenon would occur for arbitrary block sizes and for any two messages as long as the result of padding the two causes them to be of the palindromic form $A \| B \| A$ and $B \| A \| B$. Again, other non-palindromic forms such as $A \| B \| C \| B \| A$ and $B \| C \| B \| A \| B$ also cause collisions. Therefore, it is clear that extreme care has to be taken when padding messages. A check that any two messages be distinct should be done even *after* padding.

Note further that the need for padding is especially so for hashing shorter messages with the PCIHF variant in Goi et al. (2003b) that uses a large modulus, thus our concern here must be addressed with caution.

3.3. *Cyclic chaining*

The first designer of PCIHF (Goi et al., 2001, 2003a,b) personally remarked that cyclic chaining might strengthen the PCIHF construction. In particular, cyclic chaining also pairs up the last message block with the first block, i.e. $R(M[n] \| M[1])$. This is good since it seems that such a chaining prevents post-pend and pre-pend attacks so that the attacker cannot attack by using messages of arbitrary lengths because an extra constraint is put on the chaining, namely that the head and tail of the message chains have been fixed to $M[1]$. As an extra security measure, it is further suggested that for such cyclic chaining, the message length should be included as a parameter as well. However, the following observation applies even in such a case since all messages considered are of equal length.

Consider then a variant of the incremental hash function based on pair block chaining (e.g. Bellare et al., 1995a; Goi et al., 2001, 2003b) that cyclically chains the message blocks such that given a message $A \| B \| C$, the chaining is of the form:

$$\mathrm{hash}(A\|B\|C) = R(A\|B) + R(B\|C) + R(C\|A), \quad (2)$$

where $+$ denotes the combining sum operator (XOR for Bellare et al. (1995a) and modulo addition for Goi et al. (2001, 2003a,b)). Then, there is another message $B \| C \| A$ such that

$$\mathrm{hash}(B\|C\|A) = R(B\|C) + R(C\|A) + R(A\|B) \quad (3)$$

and we get the same hash output value. Also, we observe that there is yet another message $C \| A \| B$ such that

$$\mathrm{hash}(C\|A\|B) = R(C\|A) + R(A\|B) + R(B\|C) \quad (4)$$

and once again the hash output is the same.

Essentially we can think of these three messages as being 'slid' versions of each other (see Fig. 1), and since the combining function used for such schemes (whether XOR or modular addition) is commutative, they all hash to the same value. This sliding of the messages is possible since there is no block dependence in the cyclic chaining. In general, for an n-block cyclically chained message, there are n such messages that produce the same hash output when such cyclic chaining is used.

As a side remark, we note that the double-length hash function recently proposed in Nandi et al. (2005) also cyclically chains its input blocks prior to being input to a randomizing

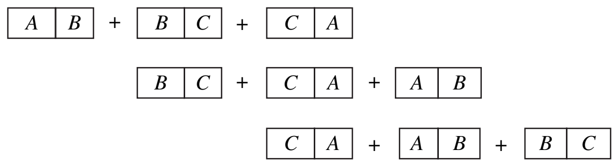

Fig. 1 – Sliding cyclically chained pair blocks.

function, $R(\cdot)$ and the outputs are then combined via XOR. However, this is not susceptible to our 'sliding' collisions since it uses a different randomizing function for every pairwise-chained block.

4. Security considerations

In this section, we first review the standard security criteria for hash functions, and then discuss the security considerations for incremental hash functions in general and also those constructed via pair block chaining. This list (though not entirely exhaustive), includes all considerations highlighted in previous work done on incremental hash functions in Bellare et al. (1994, 1995a,b, 1997), Fischlin (1997), Goi et al. (2001, 2003a,b), Wagner (2002). The three general security criteria for any hash function are:

Preimage resistance (one-wayness): for a given output y, it is computationally infeasible to find an input x such that $y = h(x)$.
Second-preimage resistance (weak collision resistance): for a given input x, it is computationally infeasible to find another input $x' \neq x$ such that $h(x) = h(x')$.
(Strong) collision resistance: it is computationally infeasible to find a pair of inputs x and $x' \neq x$ such that $h(x) = h(x')$.

Additionally, for the case of *incremental* hash functions, specifically for the *randomize-then-combine* paradigm, we have the following which are essentially consequences of the failure to provide the above-defined collision resistance properties:

(1) To prevent collisions of hash outputs due to input messages with the same but permuted blocks (of the kind we considered in Section 3), then:
   (a) Include some block dependence (the block counter, i) into the randomizing or the combining function (Bellare et al., 1997, 1995b), even if pair block chaining (Bellare et al., 1995a) is used,[2] for instance by concatenating i to the message blocks before they enter the pseudo-random function, R:

$$R(M[i]\|i). \tag{5}$$

   or
   (b) All message blocks should be distinct, even after padding.

(2) To prevent collisions found via solving linear equations as shown in Bellare et al. (1997), do not use XOR as a combining function.

With regards to consideration (1a), we note that although it has been suggested in the past (Bellare et al., 1997) to include some block dependence to a message block before it is passed to the randomizing function, R, such a consideration has not been applied to incremental hash functions based on pair block chaining such as the incremental XOR scheme (Bellare et al., 1995a) and the PCIHF (Goi et al., 2001, 2003a,b). It was always *either* block dependence *or* pair block chaining to be used, not both at the same time. We therefore stress here that this consideration is an important one and should equally apply to such incremental hash functions. Adding the block counter only slightly increases the input length of $R(\cdot)$ but otherwise does not degrade the performance. As a further remark on pair block chaining, note that instead of directly combining the randomized paired blocks into the final hash, one could use a CBC-based chaining method (NIST, 2001) where for instance the output of a previous $R(\cdot)$ is XORed to the input of the current $R(\cdot)$, etc. This requires an extra $R(\cdot)$ computation for every block updated but prevents the collisions as described in Section 3.

Consideration (1b) is often thought to be a very strong and impractical restriction. However, we argue that if this is really the case, then incremental hash functions based on pair block chaining such as in Bellare et al. (1995a), Goi et al. (2001, 2003a,b) are not themselves practical since their security and collision resistance depend on such an impractical assumption. Nevertheless, this consideration would not be necessary if consideration (1a) is incorporated.

It has been shown in Bellare et al. (1997) that the use of XOR as a combining function causes an incremental hash function based on the randomize-then-combine paradigm to fall to collisions obtained via solving linear equations. Instead, other combining functions such as modulo addition or multiplication were suggested. We remark here that this consideration alone is not sufficient against collisions of the sort presented in Section 3, e.g. PCIHF uses modulo addition for combining and yet such collisions may pose a problem. Basically, any combining function that is commutative would fail here because we need to ensure that when one or more blocks of an input message are permuted, the same hash output does not result. This highlights that the order in which the blocks are combined is important to prevent the collisions.

Finally, we reiterate that all design assumptions should be explicitly stated by the designer. Though this seems standard, it is often overlooked (Lowe, 1996) in designers' proposals, and this is a fatal mistake since most implementers are themselves not security experts and so are unfamiliar with the "standard" assumptions made by security designers. In fact, this separation of roles between design and implementation is often a problem in any area of security.

4.1. *Other incremental cryptographic primitives*

We feel it is worth mentioning as an aside the other security considerations when incrementality is applied to the case of other cryptographic primitives including digital signatures

[2] Because our results in Section 3 have shown that pair block chaining alone is insufficient to prevent collisions.

(Bellare et al., 1994), and message authentication codes (MACs) (Bellare et al., 1995a,b).

An incremental digital signature is directly derived (Bellare et al., 1994) from an incremental hash function by hashing the message with the incremental hash function to obtain μ, and then signing μ with a standard digital signature. In this case, the resistance of the scheme against existential forgery under adaptive chosen message attacks should be considered. This means that it should be computationally infeasible for an attacker to find (forge) a valid signature s for any new message m, even when he has access to a signing oracle into which he can arbitrarily submit any other message of his choice.

An incremental MAC may be viewed as a variant of the incremental hash function where the randomizing function, $R(\cdot)$ is keyed by a secret, and thus this MAC output should be computable only by legitimate parties. The security criterion then for such MACs is that an attacker should not be able to compute a valid MAC output. It was highlighted in Bellare et al. (1995b) that block dependence should be included to prevent attacks similar to our collisions in Section 3. Also, if there are messages $(m_1, m_2, \ldots)$ that have valid incremental MAC outputs $(z_1, z_2, \ldots)$, then an attacker could generate a new MAC output z' that is a valid MAC of a new message m', both obtained by linearly combining the messages and the MACs, respectively. In order to prevent this, it was suggested to use nonces (Bellare et al., 1995b) (which are random numbers used only once) in the input to $R(\cdot)$ or to use an extra (Bellare et al., 1995a) nonlinear pseudo-random permutation $R_2(\cdot)$ at the output of the combining function.

5. Concluding remarks

Incremental hash functions (Bellare et al., 1994) are good primitives compared to conventional non-incremental ones, but care has to be taken when designing them, especially to avoid collisions. We have shown three cases of how collisions can occur in incremental hash functions based on pair block chaining. In particular, this shows that the incremental XOR scheme (Bellare et al., 1995a) is not collision-resistant, and that the PCIHF (Goi et al., 2001, 2003a,b) would equally fail to be collision-resistant if it is not implemented carefully in several settings. Our results highlight that extreme caution should be taken by hash function designers when designing such hash functions, and that whatever assumptions made should be stated explicitly so that they will be noted by hash function implementers. As a first step towards concrete incremental hash function design, we highlighted some security considerations for designing and implementing collision-resistant incremental hash functions based on pair block chaining.

Acknowledgement

The first author is grateful to B.-M. Goi for helpful discussions on the PCIHF and comments on some initial ideas presented here. We also thank the anonymous referees for their many constructive comments and suggestions that have significantly improved this paper.

REFERENCES

Bellare M, Goldreich O, Goldwasser S. Incremental cryptography: the case for hashing and signing. In: Advances in cryptology – Crypto '94. Lecture notes in computer science, vol. 839. Germany: Springer-Verlag; 1994. p. 216–33.

Bellare M, Goldreich O, Goldwasser S. Incremental cryptography and application to virus protection. In: Proceedings of 27th ACM symposium on the theory of computing (STOC); 1995a. p. 45–56.

Bellare M, Guerin R, Rogaway P. XOR MACs: new methods for message authentication using finite pseudorandom functions. In: Advances in cryptology – Crypto '95. Lecture notes in computer science, vol. 963. Germany: Springer-Verlag; 1995b. p. 15–29.

Bellare M, Goldreich O, Goldwasser S. A new paradigm for collision-free hashing: incrementality at reduced cost. In: Advances in cryptology – Eurocrypt '97. Lecture notes in computer science, vol. 1233. Germany: Springer-Verlag; 1997. p. 163–92.

Fischlin M. Incremental cryptography and memory checkers. In: Advances in cryptology – Eurocrypt '97. Lecture notes in computer science, vol. 1233. Germany: Springer-Verlag; 1997. p. 393–408.

Goi B-M, Siddiqi MU, Chuah H-T. Incremental hash function based on pair chaining & modular arithmetic combining. In: Progress in cryptology – Indocrypt '01. Lecture notes in computer science, vol. 2247. Germany: Springer-Verlag; 2001. p. 50–61.

Goi B-M, Siddiqi MU, Chuah H-T. An attack strategy for incremental hash function based on pair chaining & XOR combining. In: Proceedings of international conference on applied cryptography and network security (ACNS '03), Industrial track; 2003a.

Goi B-M, Siddiqi MU, Chuah H-T. Computational complexity and implementation aspects of the incremental hash function. IEEE Transactions on Consumer Electronics 2003b;49(4): 1249–55.

Lowe G. Some new attacks upon security protocols. In: Proceedings of ninth IEEE computer security foundations workshop; 1996.

Nandi M, Lee W, Sakurai K, Lee S. Security analysis of a 2/3-rate double length compression function in black-box model. In: Fast software encryption (FSE '05). Lecture notes in computer science, vol. 3557. Germany: Springer-Verlag; 2005. p. 243–54.

NIST. Recommendation for block cipher modes of operation – methods and techniques, NIST Special Publication, SP 800-38A; 2001.

Wagner D. A generalized birthday problem. In: Advances in cryptology – Crypto '02. Lecture notes in computer science, vol. 2442. Germany: Springer-Verlag; 2002. p. 288–303.

Raphael C.-W. Phan is currently the Director of the Information Security Research (iSECURES) Laboratory at the Swinburne University of Technology (Sarawak Campus) – SUTS, Kuching, Malaysia. Raphael researches on cryptography, cryptanalysis, authentication and key exchange protocols, smart card security, hash functions and digital watermarking. His work has been published in refereed journals published by IEE, IEEE, Elsevier Science and US Military Academy, and in internationally refereed cryptology conferences published by Springer-Verlag, Germany. He is referee for several IEEE journals on the area of information security. He is General Chair of Mycrypt '05 and Asiacrypt '07, Program Chair of the International Workshop on Information Security & Hiding (ISH '05), and technical Program Committee member of Mycrypt '05, the International Conference on Information Security & Cryptology (ICISC '05) and International Conference on Applied Cryptography & Network Security (ACNS '06).

David Wagner is an Assistant Professor in the Computer Science Division at the University of California at Berkeley with extensive experience in computer security and cryptography. He and his Berkeley colleagues are known for discovering a wide variety of security vulnerabilities in various cellphone standards, 802.11 wireless networks, and other widely deployed systems. In addition, David was a co-designer of one of the Advanced Encryption Standard candidates, and he remains active in the areas of systems security, cryptography, and privacy.

David is an Alfred P. Sloan Research Fellow and a CRA Digital Government Fellow. He received an Honorable Mention in the ACM Doctoral Dissertation Award competition for his Ph.D. work.