Incremental Cryptography and Memory Checkers

Marc Fischlin*
Fachbereich Mathematik/Informatik
Johann Wolfgang Goethe-Universität Frankfurt am Main
PSF 111932
60054 Frankfurt/Main, Germany
e-mail: marc@informatik.uni-frankfurt.de

Abstract

We introduce the relationship between incremental cryptography and memory checkers. We present an incremental message authentication scheme based on the XOR MACs which supports insertion, deletion and other single block operations. Our scheme takes only a constant number of pseudorandom function evaluations for each update step and produces smaller authentication codes than the tree scheme presented in [BGG95]. Furthermore, it is secure against message substitution attacks, where the adversary is allowed to tamper messages before update steps, making it applicable to virus protection. From this scheme we derive memory checkers for data structures based on lists. Conversely, we use a lower bound for memory checkers to show that so-called message substitution detecting schemes produce signatures or authentication codes with size proportional to the message length.

1 Introduction

The notion of incremental cryptography has been introduced by Bellare, Goldreich and Goldwasser in [BGG94] and refined by the same authors in [BGG95]. Suppose that we are given a block-by-block message M and its cryptographic form μ, i.e. encryption, signature or authentication code. Let M' be a message that is obtained by applying a text modification from a set $\mathcal{M}$ of modifications to M. With an incremental scheme supporting the text modifications $\mathcal{M}$ a cryptographic form μ' for M' can be produced much faster from μ and M than it would take to compute it from scratch.

Our results. We present the incremental authentication scheme IncXMACC that supports single block insertion and deletion, and therefore other operations like replacement. To update an authentication code for inserting or deleting a single block at a given position, this scheme performs only a constant number of pseudorandom function evaluations. Additionally, insertion can be done without accessing the message and deletion merely needs the corresponding block.

Security against Message Substitution Attacks. Our scheme remains secure if an adversary is allowed to alter messages before applying the update algorithm — while the shorter authentication code must be kept on some secure medium. Security against these message substitution attacks implies application to virus protection. To protect a large file stored on some insecure medium

*URL: http://www.uni-frankfurt.de/~roessner/group/marc/marc.html

against unauthorized alternation, authenticate this file and store the shorter authentication code in some incorruptible memory. Whenever an authorized user modifies the file, we can update the authentication code very fast using the incremental algorithm. Conversely, it is very unlikely that an attacker, e.g. a virus, will be able to produce a forgery even if he tampers the documents before update steps. In this sense, message substitution attacks lie between (total) substitution attacks, where both the message and signature can be tampered before update steps, and basic attacks, where the adversary isn't allowed to alter messages or signatures before updating.

Related Work. In [BGG94] a hash-and-sign scheme based on an incremental hash function was presented. The signature consists of the hash value h and a signature for h produced by an arbitrary non-incremental signature scheme. To update a signature, increment the hash value and sign this new hash value. Unfortunately, this scheme only supports single block replacement and it is provably not secure against message substitution attacks.

In [BGG95] the same authors present the tree scheme supporting single block operations like insertion and deletion (and the more powerful modifications cut and paste to devide a text into two documents resp. to append a document to another). The tree scheme takes $\Omega(\log n)$ verification and authentication steps for the abovementioned operations, where n is the number of blocks of the document. For the cut modification, the tree scheme is much faster than IncXMACC, while our scheme supports the insert, delete and paste modifications applying a pseudorandom function only a constant number of times. Moreover, our scheme produces considerably smaller authentication codes than the tree scheme, though the authentication code must be kept on a secure medium. In contrast to that, signatures and authentication codes produced by the tree scheme can be stored in the insecure memory. A randomized version of the tree scheme is given in [M97]. This scheme hides the fact whether the incremental or non-incremental algorithm has been used to produce a signature.

Our scheme IncXMACC refines the incremental authentication scheme presented in [BGG95], which is also based on the XOR MACs. This scheme has several disadvantages in comparison to our scheme: It doubles the key size by using two pseudorandom functions and it requires many random bits. For an update step the incremental algorithm reads more than the corresponding block and security has only been proven for basic attacks.

Memory Checkers. Using IncXMACC, we present a method to obtain memory checkers for lists and similar data structures. Informally, a memory checker for a data structure $\mathcal{D}$ verifies that for a given sequence of operations, an implementation of $\mathcal{D}$ works correctly for this sequence. If not, the checker outputs some error message. There are two sources of errors: The program implementing the data structure can be buggy or the memory where the elements are stored can be tampered by an adversary, e.g. a virus. Intuitively, incremental schemes that are secure against message substitution attacks seem to provide a suitable method to design such checkers. To do so, keep a signature for the current memory content and update the signature accordingly for an operation for $\mathcal{D}$. Nevertheless, in some settings the checker should be able to update the signature given only the old signature and the element resp. block that for example shall be deleted or inserted, without accessing other parts of the memory content. IncXMACC has this property.

Making the connection between memory checkers and incremental schemes we transfer a lower bound for checkers to incremental schemes. Informally, an incremental scheme is message substitution detecting, if it detects when relevant parts of message have been altered before calling the update algorithm. We give a sufficient condition under which an incremental message substitution detecting scheme that is secure against basic attacks, is also secure against message substitution attacks. The lower bound states that the length of a signature produced by a substitution detecting scheme must be very large, roughly proportional to the size of the message.

For a discussion about the differences between the memory checker setting and the program checking model (which has been introduced by Blum and Kannan in [BK89]) resp. the software protection model of Goldreich and Ostrovsky [GO96] we refer the reader to [BEG+94].

EXACT SECURITY. We follow the paradigm presenting our results in terms of *exact security* [BKR94, BGR95]. Informally, the notion of exact security can be described as follows. Assume that we have an adversary for IncXMACC with running time[1] t that makes at most q signature queries for messages of length at most L and achieves success probability ϵ. Then we derive (in a constructive way) a distinguisher D for the underlying function family F with parameters t', q', ϵ', such that D can distinguish F and the family of all functions with running time t', making at most q' oracle queries and achieving advantage at least ϵ'. Here, t', q', ϵ' are determined by t, q, L, ϵ.

OUTLINE. In section 2 we review the definition of incremental schemes from [BGG95]. In section 3 we present our incremental message authentication scheme. Section 4 deals with the relationship of incremental schemes and memory checkers.

2 Incremental Cryptography

We briefly review the definitions of incremental cryptography. This part is mainly based on [BGG95]. See this work for further discussion. In section 2.2 we introduce the notion of message substitution attacks.

2.1 Incremental Schemes

Let $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Vf})$ be an ordinary (i.e. non-incremental) signature or message authentication scheme which allows to sign block messages. That is, on input a security parameter s and a block size b in unary, the Gen algorithm outputs in probabilistic polynomial time a pair of keys (e, d). For simplicity we assume that s and b are recoverable from e or d and that $b = \mathrm{poly}(s)$. On input the key d and an admissible message $M \in \Sigma^*$, where $\Sigma = \{0,1\}^b$, the signer Sig outputs a signature or message authentication code (MAC) μ in probabilistic polynomial time in s (and b). The polynomial time verifier Vf outputs a bit a where $a = 1$ stands for "accept" and $a = 0$ for "reject". A scheme is called *complete*, if $\mathsf{Vf}(e, M, \mathsf{Sig}(d, M)) = 1$ for all keys produced with positive probability by Gen and all admissible messages M. We say that a signature μ for M is *valid*, if $\mathsf{Vf}(e, M, \mu) = 1$. Else it is called *invalid*.

To every document we associate a name $\alpha \in \{0,1\}^*$ and a counter cnt_α. For the rest of this paper, we assume that the counter value is bounded above by 2^b and that the document name has length at most b, so that both values can be treated as message blocks, and that all messages $M \in \Sigma^i$ with $1 \le i \le \mathrm{poly}(s)$ are admissible. Let $\pi(M_1, \ldots, M_m, y) \in \Sigma^*$ denote the message that is obtained by applying text modification π to messages $M_1, \ldots, M_m$ with argument vector y. For example, $\pi(M, i, M_*) = \mathsf{replace}(M, i, M_*)$ for $y = (i, M_*)$ is the message where the i^{th} block in M is replaced by $M_* \in \Sigma$. We only present the definition for incremental signature schemes. The definition for message authentication schemes is similar.

Definition 2.1 (Incremental Signature Scheme)
Let $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Vf})$ be a signature scheme and $\mathcal{M}$ a set of text modifications. An $\mathcal{M}$-incremental scheme is an interactive machine such that:

[1] To be precise, t describes the running time *and* the size of the adversary's algorithm. For simplicity, we will only deal with the issue of running time in this paper.

- *The machine is initialized with a pair* (e, d) *of keys produced by* Gen *on input* $(1^s, 1^b)$.
- *For a* create *command with arguments* $\alpha \in \{0,1\}^*$ *and* $D \in \Sigma^*$, *the machine initializes a counter* cnt_α *with* 1 *and produces a signature* $\mathsf{Sig}(d, D)$. *(The* Sig *algorithm might take as additional input the name* α *and* cnt_α.*) The machine stores the document* D, *the counter* cnt_α *and the signature with reference to name* α. *If a document for this name already exists, it is replaced by* D *and* cnt_α *is incremented instead of initialized before calling* Sig.
- *On an* edit *command for the text modification* $\pi \in \mathcal{M}$ *with argument vector* y *and document names* $\alpha_1, \ldots, \alpha_m$ *and* β, *the machine works as follows:*
  - *The machine increments the counter of document* β.
  - *It updates the signature of the document for* β.
  - *It replaces the document specified by* β *by applying modification* π *with argument vector* y *to the documents defined by the values* α_i.

  The update step is done by applying the incremental algorithm IncSig *to the documents* D_{α_i} *and signatures* μ_{α_i} *specified by the values* α_i, *the modification* π *with argument vector* y *and the key* d.[2] *We write* $\mathsf{IncSig}(d, D_{\alpha_1}, \ldots, D_{\alpha_m}, \mu_{\alpha_1}, \ldots, \mu_{\alpha_m}, \pi, y)$ *for the probabilistic output that describes the signature for the document defined by* β. *The algorithm might take as additional input all the counter values and document names, including* β *and* cnt_β.

The incremental scheme is called complete, if $\mathcal{S}$ *is complete and for all pairs* (e, d) *of keys which are produced by* Gen *with positive probability and all valid signatures* μ_{α_i} *for* D_{α_i}, *the output of* IncSig *satisfies* $\mathsf{Vf}(e, D_\beta, \mathsf{IncSig}(d, D_{\alpha_1}, \ldots, D_{\alpha_m}, \mu_{\alpha_1}, \ldots, \mu_{\alpha_m}, \pi, y)) = 1$, *where the verifier* Vf *might take* β *and* cnt_β *as additional input.*

For simplicity, we also write $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{IncSig}, \mathsf{Vf})$ for the incremental scheme. Additionally, let $\mathcal{S}(b, s)$ denote the incremental scheme with fixed parameters b and s.

2.2 Security

In this section we review the notion of security for incremental signature and authentication schemes. Basically, an adversary performs an adaptive chosen message attack [GMR88]. So far, all values are stored securely by the interactive machine. As done in [BGG95], we augment our model by an alter command that takes as arguments a document name α, a document $D \in \Sigma^*$ and a signature μ. For an alter command the interactive machine replaces the document with name α by D and the signature by μ regardless of the current values. The counter value cnt_α remains unchanged.

The alter command models the following settings: Suppose that the documents and signatures are kept on an insecure medium like a remote host. Then an adversary, e.g. a virus, might change the document before issuing an edit command. If the adversary doesn't use alter commands during his attack, we call it a *basic attack*. If he tampers only documents but no signatures, we call this a *message substitution attack*. This corresponds to the case when the possibly short signature is kept on a secure medium. If the adversary changes documents and signatures, it is called a *(total) substitution attack*.

In substitution attacks, we must associate the signature or authentication code to some document. [BGG95] therefore introduce *virtual documents*. To every document D we define the

[2] To be more precise, the IncSig algorithm is passed a description of π. We assume that $|\mathcal{M}|$ is constant.

virtual document $\mathrm{virt}(D)$ as follows: If the document D was issued by a create command, let $\mathrm{virt}(D) = D$. If the document was obtained by an edit command applying π with argument vector y to documents $D'_1, \ldots, D'_m$, let $\mathrm{virt}(D)$ be the document that is obtained by applying π with y to $\mathrm{virt}(D'_1). \ldots, \mathrm{virt}(D'_m)$. If the document D was obtained by an alter command replacing document D', let $\mathrm{virt}(D) = \mathrm{virt}(D')$. An adversary is *successful*, if he produces a signature or authentication code for a document which hasn't appeared as a virtual document before. We define security in terms of exact security:

Definition 2.2 (Security of Incremental Schemes)
Let $\mathcal{S}(b, s)$ be an incremental signature or message authentication scheme with block size b and security parameter s. A $(t, q_s, q_v, q_i, L_s, L_v, L_i, \epsilon)$-adversary E makes at most t steps (in a standard RAM model [AHU74]), queries Sig, IncSig, Vf *at most q_s, q_i, q_v times, each query with messages of no more than L_s, L_i, L_v blocks, and is successful with probability at least ϵ. $\mathcal{S}(b, s)$ is said to be $(t, q_s, q_v, q_i, L_s, L_v, L_i, \epsilon)$-secure against basic/message substitution/total substitution attacks, iff there is no $(t, q_s, q_v, q_i, L_s, L_v, L_i, \epsilon)$-adversary performing the corresponding attack.*

For the rest of this paper, we write $(t, \vec{q}, \vec{L}, \epsilon)$ for $\vec{q} = (q_s, q_i, q_v)$ and $\vec{L} = (L_s, L_i, L_v)$. In some settings, parameters may be irrelevant, for example q_v and L_v in signature schemes. It this case, it is understood that $\vec{q}$ and $\vec{L}$ abbreviate (q_s, q_i) and (L_s, L_i).

3 Incremental Message Authentication: IncXMACC

In this section we present the stateful incremental message authentication scheme IncXMACC.

3.1 Notations and Definitions

For two strings $x, y \in \{0,1\}^*$, let $x \cdot y$ be the concatenation of x and y. For $x, y \in \{0,1\}^n$, $x \oplus y$ denotes the bitwise exclusive-or of x, y. For a number $i \in \{0, \ldots, 2^m - 1\}$, let $\langle i \rangle_m$ denote the m-bit binary representation of i.

Let $\mathrm{Map}(X, Y)$ denote the set of all functions with domain X and range Y. A *function family* $F \subseteq \mathrm{Map}(X, Y)$ is a set of functions, where we associate a key a to each function $f \in F$. Let F_a be the function specified by key a. To draw a function $f \in F$ at random means to choose at random with equal probability a key a from the set of all keys of functions in F and to set $f := F_a$. For a function f from the family $\mathrm{Map}(X, Y)$ the associated key is the sequence of all $|X|$ function values in some fixed order.

Let $F, G \subseteq \mathrm{Map}(X, Y)$ be two function families and D be a probabilistic algorithm. Define the *advantage* of D distinguishing between F and G as

$$\mathrm{Adv}_D(F, G) = \mathrm{Prob}_{f \in F}\left[D^f = 1\right] - \mathrm{Prob}_{g \in G}\left[D^g = 1\right],$$

where the probabilities are taken over the random choice of $f \in F$ resp. $g \in G$ and the coin tosses of D. We say that D is a (t, q, ϵ)-*distinguisher* if it makes at most t steps (in a standard RAM model), makes at most q oracle queries and achieves $\mathrm{Adv}_D(F, \mathrm{Map}(X, Y)) \geq \epsilon$. We say that the family F is (t, q, ϵ)-*secure* if there exists no (t, q, ϵ)-distinguisher.

3.2 XOR Schemes

Bellare, Guérin and Rogaway [BGR95] introduced the XOR MAC schemes, a general framework for designing message authentication schemes. Let F be a function family with domain $\{0,1\}^l$ and range $\{0,1\}^L$ and let F_a be a function in F according to key a. Given a message $M = M[1]\cdots M[n]$ and some state information, e.g. a counter, an algorithm $\mathcal{R}$ outputs probabilistically some seed r. On input r and M, a deterministic algorithm $\mathcal{E}$ produces a set $Z \subseteq \{0,1\}^l$. Both algorithms must not dependend on the key a. The message authentication code for M is (r, z), where $z = \bigoplus_{x\in Z} F_a(x)$. The verifier knowing the key a works as follows: On input a MAC (r', z') and a message M', it runs $\mathcal{E}$ with input r' and M' to obtain a set $Z' \subseteq \{0,1\}^l$ and accepts iff $\bigoplus_{x\in Z'} F_a(x) = z'$.

Security of such schemes can be reduced to the algebraic problem that an associated matrix has full rank. For a set $Z \subseteq \{0,1\}^l$ let the characteristic 2^l-bit vector be the vector where the x^{th} entry is 1 iff $x \in Z$. Assume that the underlying function family is $\mathrm{Map}(\{0,1\}^l, \{0,1\}^L)$. Then the probability that the verifier accepts one of the q_v queries for a new message is bounded above by $\delta := q_v \cdot 2^{-L} + \max_{M,r}\{\mathsf{NFRank}_{q_s}(M, r)\}$ with

$$\mathsf{NFRank}_{q_s}(M, r) := \mathrm{Prob}[\,\mathsf{Matrix}_{q_s}(M, r) \text{ hasn't full rank} \mid M \notin \{\mathsf{M_1}, \ldots, \mathsf{M_{q_s}}\}\,]$$

Here, $\mathsf{Matrix}_{q_s}(M, r)$ describes the random matrix over GF[2], consisting of the $q_s + 1$ characteristic vectors, where the first q_s vectors for the signing queries are defined by $\mathcal{E}$'s output for the random messages $\mathsf{M_i}$ and seeds $\mathsf{R_i}$, and the row vector $q_s + 1$ is specified by $\mathcal{E}$'s output for the possible forgery M and seed r in the first verify query. Note that these two values determine the MAC, since $\mathcal{E}$ is deterministic. Given an adversary A for such an XOR scheme based on a function family F such that A is successful with probability ϵ', one can derive a distinguisher for F with comparable running time and advantage $\epsilon \geq \epsilon' - \delta$. See for example [BGR95] or the proof of Theorem 3.2.

3.3 The Scheme IncXMACC

The scheme $\mathsf{IncXMACC}_{F,b}$ is based on a function family $F \subseteq \mathrm{Map}(\{0,1\}^l, \{0,1\}^L)$ and has block size $b \leq l^*$ where $l^* = \frac{1}{2}l - 1$. For notational convenience we assume that l is even. It supports the operations $\mathsf{insert}(M, i, M_*)$ and $\mathsf{delete}(M, j)$ for inserting block M_* at position i resp. deleting the j^{th} block in message $M = M[1]\cdots M[n]$, where $1 \leq i \leq n+1$ and $1 \leq j \leq n$. Therefore, the scheme supports other operations like $\mathsf{replace}(M, i, M_*)$, $\mathsf{swap}(M, i, j)$ or $\mathsf{move}(M, i, j)$ to replace block i by M_*, to swap block i and j or to move block i to position j, respectively. We sometimes abbreviate $\mathsf{delete}(M, j)$ by $\mathsf{delete}(j)$ if the corresponding message M is clear from the context. Similar for the other operations.

We will first discuss the single document setting and then show how to proceed in the multi document case. In the single document model, the scheme holds two counters dcnt and bcnt, a document counter resp. a block counter, both initialized with 0. For technical reasons, only messages with more than two blocks are allowed. In the multi document setting, only message with more than four blocks are admissible. In both cases, the counter values are bounded above by 2^{l^*}. The underlying idea is that we link every message block to a unique block counter value and incorperate the order of the message blocks by chaining the counter values.

We define the algorithms Sig and IncSig. Assume that the user or adversary issues a create command for the document $M[1]\cdots M[n] \in \Sigma^n$. Then Sig increments dcnt by one and produces the

MAC $\big(\mathrm{dcnt}, \mathrm{bcnt}+1, \ldots, \mathrm{bcnt}+n, z\big)$, where $z = \bigoplus_{x \in Z} F_a(x)$ with

$$Z = \{0 \cdot \langle \mathrm{dcnt} \rangle_{l-1}\} \cup \{10 \cdot \langle M[i] \rangle_{l^*} \cdot \langle \mathrm{bcnt}+i \rangle_{l^*} \mid i = 1, \ldots, n\}$$
$$\cup \{11 \cdot \langle \mathrm{bcnt}+i \rangle_{l^*} \cdot \langle \mathrm{bcnt}+i+1 \rangle_{l^*} \mid i = 1, \ldots, n-1\}$$

Finally, Sig increments bcnt by n. On an $\mathsf{insert}(i, M_*)$ command for the current document $M = M[1] \cdots M[n]$ and MAC $\mu = (d, c_1, \ldots, c_n, z)$ for M, the system works as follows: IncSig increments the counters dcnt and bcnt and outputs a new MAC $\big(\mathrm{dcnt}, c_1, \ldots, c_{i-1}, \mathrm{bcnt}, c_i, \ldots, c_n, z'\big)$ for the document $M[1] \ldots M[i-1]\, M_*\, M[i] \cdots M[n]$, where

$$z' = z \oplus F_a(0 \cdot \langle d \rangle_{l-1}) \oplus F_a(0 \cdot \langle \mathrm{dcnt} \rangle_{l-1}) \oplus F_a(10 \cdot \langle M_* \rangle_{l^*} \cdot \langle \mathrm{bcnt} \rangle_{l^*})$$
$$\oplus F_a(11 \cdot \langle c_{i-1} \rangle_{l^*} \cdot \langle c_i \rangle_{l^*}) \oplus F_a(11 \cdot \langle c_{i-1} \rangle_{l^*} \cdot \langle \mathrm{bcnt} \rangle_{l^*}) \oplus F_a(11 \cdot \langle \mathrm{bcnt} \rangle_{l^*} \cdot \langle c_i \rangle_{l^*})$$

That is, the old document counter value of the document is replaced by the new one and the new block M_* is linked to its block counter value bcnt (first line). Moreover, bcnt is put in the chain between c_{i-1} and c_i breaking up the link between c_{i-1} and c_i (second line). In case of $i = 1$ (resp. $i = n+1$) the first and second (resp. third) function value of the last line is left out.

A $\mathsf{delete}(i)$ command for $1 \le i \le n$ is processed similarly. Having incremented dcnt, the new MAC for the document $M[1] \cdots M[i-1] M[i+1] \cdots M[n]$ is given by $\big(\mathrm{dcnt}, c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n, z'\big)$ where

$$z' = z \oplus F_a(0 \cdot \langle d \rangle_{l-1}) \oplus F_a(0 \cdot \langle \mathrm{dcnt} \rangle_{l-1}) \oplus F_a(10 \cdot \langle M[i] \rangle_{l^*} \cdot \langle c_i \rangle_{l^*})$$
$$\oplus F_a(11 \cdot \langle c_{i-1} \rangle_{l^*} \cdot \langle c_i \rangle_{l^*}) \oplus F_a(11 \cdot \langle c_i \rangle_{l^*} \cdot \langle c_{i+1} \rangle_{l^*}) \oplus F_a(11 \cdot \langle c_{i-1} \rangle_{l^*} \cdot \langle c_{i+1} \rangle_{l^*})$$

In this case, the system doesn't increment bcnt. For $i = 1$ or $i = n$ adapt the last line as above.

Finally, we define the verify procedure Vf. Given a document $M = M[1] \ldots M[n]$ and a MAC $(d', c'_1, \ldots, c'_{n'}, z')$, check that $n' = n$ and that all r'_j values are different and reject if one of these properties doesn't hold. Otherwise compute $z = \bigoplus_{x \in Z} F_a(x)$ with

$$Z = \{0 \cdot \langle d' \rangle_{l-1}\} \cup \{10 \cdot \langle M[i] \rangle_{l^*} \cdot \langle c'_i \rangle_{l^*} \mid i = 1, \ldots, n\}$$
$$\cup \left\{11 \cdot \langle c'_i \rangle_{l^*} \cdot \left\langle c'_{i+1} \right\rangle_{l^*} \;\middle|\; i = 1, \ldots, n-1\right\}$$

Reject if $z \neq z'$, otherwise accept.

Security is proven as in [BGR95]. We first deal with the case $F = R = \mathrm{Map}(\{0,1\}^l, \{0,1\}^L)$ and show an upper bound for the success probability. A sketch of the proof is given in Appendix A.

Theorem 3.1
Let $R = \mathrm{Map}(\{0,1\}^l, \{0,1\}^L)$ and $2b + 2 \le l$. Let E be a computationally unbounded adversary attacking the incremental scheme $\mathsf{IncXMACC}_{R,b}$ in a message substitution attack making at most q_v verify queries. The probability that E is successful is bounded above by $\delta_I := q_v \cdot 2^{-L}$.

Obviously, this bound is tight. From this Theorem we derive:

Theorem 3.2
Let $F \subseteq \mathrm{Map}(\{0,1\}^l, \{0,1\}^L)$ be a function family with $2b + 2 \le l$. If F is (t', q', ϵ')-secure then $\mathsf{IncXMACC}_{F,b}$ is $(t, q_s, q_v, q_e, L_s, L_v, \epsilon)$-secure, where

$$t' = t + c(q_s + q_v + q_e)(L + l + b), \quad q' = 2q_v L_v + 2q_s L_s + 6q_e, \quad \epsilon' = \epsilon - q_v \cdot 2^{-L}$$

for a small constant $c \in \mathbb{N}$ depending only on the computational model.

Proof (Sketch). Let E be an adversary for IncXMACC with the specified parameters and success probability at least ϵ. From E we construct a distinguisher D for F. D is given oracle access to a randomly chosen function g in F resp. R. D simulates E and IncXMACC's program by replacing each function evaluation F_a with the oracle values for g and outputs 1 iff E is successful. By Theorem 3.1, for $g \in R$ the adversary E is successful with probability at most $q_v \cdot 2^{-L}$. Therefore,

$$\begin{aligned}\mathrm{Adv}_D(F,R) &= \mathrm{Prob}_{g\in F}[D^g = 1] - \mathrm{Prob}_{g\in R}[D^g = 1] \\ &= \mathrm{Prob}_{g\in F}[E \text{ is successful}] - \mathrm{Prob}_{g\in R}[E \text{ is successful}] \geq \epsilon - q_v \cdot 2^{-L}.\end{aligned}$$

Hence, D is a (t', q', ϵ')-distinguisher for F. ■

We compare IncXMACC and the tree scheme presented in [BGG95]. Our scheme is only secure when the MAC is kept on a secure medium, while the tree scheme is secure against total substitution attacks. The tree scheme can be applied with any secure signature or authentication scheme, but deleting or inserting a block takes $\Omega(\log n)$ evaluations of the ordinary signature scheme, where n is the number of message blocks of the document. Additionally the tree structure must be maintained. Nevertheless, the tree scheme supports the more powerful modifications paste and cut. The advantage of our scheme is that it takes only a constant number of function evaluations for insert and delete (below we'll show that this holds also for the paste modification), that it merely accesses the corresponding message block in update steps, and that the size of the MAC is considerably smaller. Namely, let s be the output length of the pseudorandom function used by IncXMACC and the output length of the ordinary authentication scheme used in the tree scheme. Moreover, assume that both schemes have block size b. If the block counter is bounded above by s^c, then IncXMACC produces MACs for messages of n blocks with bit size at most $s + c(n+1)\log s = \mathcal{O}(s + n\log s)$, while MACs produced by the tree scheme have size at least $(\frac{3}{2}s + 1)n = \Omega(ns)$.

The scheme IncXMACC is provably not secure against (nonadaptive) total substitution attacks. The adversary queries Sig for the document ABCD, where A,B,C,D are different blocks in $\{0,1\}^b$. He alters the document to AABC and changes the MAC $(d, c_1, c_2, c_3, c_4, z)$ to $(d, c_1, c_1, c_2, c_3, z)$. Then he asks IncSig to delete the third symbol. Replacing this MAC $(d{+}1, c_1, c_1, c_3, z')$ by $(d{+}1, c_1, c_3, c_4, z')$, he obtains a valid MAC for the document ACD, which hasn't appeared as a virtual document.

We now adress the multi document setting. For every document we associate a name $\alpha \in \{0,1\}^b$. Additionaly, we keep a block counter bcnt_α and a document counter dcnt_α for each document. Signing a document is similar to IncXMACC but we use the value $00 \cdot \langle \mathrm{dcnt}_\alpha \rangle_{l^*} \cdot \langle \alpha \rangle_{l^*}$ instead of $0 \cdot \langle \mathrm{dcnt} \rangle_{l-1}$ for the source and $00 \cdot \langle \mathrm{dcnt}_\beta + 1 \rangle_{l^*} \cdot \langle \beta \rangle_{l^*}$ instead of $0 \cdot \langle \mathrm{dcnt} + 1 \rangle_{l-1}$ for the destination. Security follows as in Theorem 3.1 and Theorem 3.2.

Theorem 3.3
Let $F \subseteq \mathrm{Map}(\{0,1\}^l, \{0,1\}^L)$ be a function family with $2b + 2 \leq l$. If F is (t', q', ϵ')-secure then $\mathsf{IncXMACC}_{F,b}$ is $(t, q_s, q_v, q_e, L_s, L_v, \epsilon)$-secure in the multi document setting with at most I documents, where

$$t' = t + cI(q_s + q_v + q_e)(L + l + b), \quad q' = 2q_vL_v + 2q_sL_s + 6q_e, \quad \epsilon' = \epsilon - q_v \cdot 2^{-L}$$

for a small constant $c \in \mathbb{N}$.

In the multi document setting, we can allow a paste modification if we use *one* block counter for *all* documents. The paste command for documents M, M' with names α_1, α_2 and MACs

$(d, c_1, \ldots, c_n, z)$, $(d', c'_1, \ldots, c'_{n'}, z')$ produces the MAC $(\mathrm{dcnt}_\beta + 1, c_1, \ldots, c_n, c'_1, \ldots, c'_{n'}, \hat{z})$ with

$$\hat{z} = z \oplus z' \oplus F_a(00 \cdot \langle \mathrm{dcnt}_\beta + 1 \rangle_{l^*} \cdot \langle \beta \rangle_{l^*}) \oplus F_a(00 \cdot \langle d \rangle_{l^*} \cdot \langle \alpha_1 \rangle_{l^*}) \\ \oplus F_a(00 \cdot \langle d' \rangle_{l^*} \cdot \langle \alpha_2 \rangle_{l^*}) \oplus F_a(11 \cdot \langle c_n \rangle_{l^*} \cdot \langle c'_1 \rangle_{l^*})$$

for the document $M \cdot M'$ with name β.

4 Memory Checkers

Blum et al. introduced the notion of memory checkers [BEG+94]. In this section we refine their definition and formalize the memory checker model. From IncXMACC we derive memory checkers for lists and similar data structures. Finally, we transfer a lower bound for memory checkers to message substitution detecting schemes.

4.1 Definition

Let $\mathcal{D}$ be a data structure with a set of operations that define the behaviour of $\mathcal{D}$ on an initial configuration. Consider for example the data structure stack. The sequence $\mathsf{push}(a)$, $\mathsf{push}(b)$, pop, $\mathsf{push}(b)$, pop for an empty stack produces the output $-$, $-$, b, $-$, b, where $-$ stands for "no output".

We assume that all arguments for the operations are specified by a parameter n. To emphasize this dependence we write $\mathcal{D}_n$. We want to design a program C that checks whether an implementation D_n of $\mathcal{D}_n$ works correctly for a sequence of operations for this data structure. We call these operations *user* or *input operations*. C filters the interaction between the user and the data structure resp. memory, so that the user can interact with the data structure only via the checker. See figure 1. After having read the next user operation, the program C shall return the output of that operation to the user or BUGGY if an error occurs, e.g. D_n returns a different value than the expected one. Obviously, the worst case occurs if the user and the memory is totally under control of one adversary. Additionally, the adversary works adaptively, i.e. his next action depends on all previous steps.

To allow multiple instances, we extend every operation by an argument taking values between 0 and $I - 1$ in binary, where I stands for the maximal number of instances available. Let $\mathcal{D}_n^I$ be the augmented version of $\mathcal{D}_n$. The checker can use further instances to save additional information like time stamps to the insecure medium.

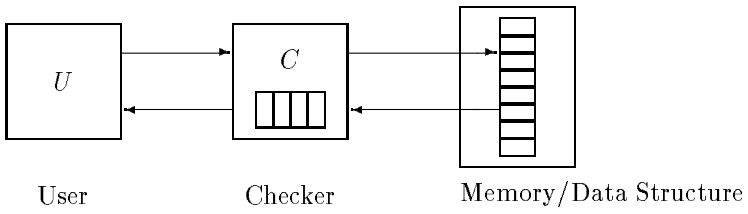

Figure 1: The memory checker model

An execution is divided into rounds. Each round starts with the checker reading the next user operation. Then it performs some local computation and may interact arbitrarily with the data structure. After having finished this computation, the checker shall return the correct answer for the user operation to the user (or "–" if the operation doesn't produce an output) before reading

the next operation. The checker shall output BUGGY if the data structure returns a faulty value at some point in the execution. On the other hand, it shall never output BUGGY if no error occurs. Before starting the first round, the checker might perform a preprocessing, and additionally, after having read the last user operation, it might do some "postprocessing" (and perhaps output BUGGY then).

We use the RAM model to define our checker. The space complexity is measured logarithmically, while time complexity can either be uniform or logarithmic. In this work, time will be meassured uniformly. We assume that the adversary's model of computation is a RAM, too, and that both RAM share a sufficient large number of registers to exchange information, while every other memory of each machine is private. See [GMR89, GO96] for a more formal treatment of interactive machines.

Definition 4.1 (Memory Checker)
A $(t_{pre}, t_{post}, t_{op}, s, q, J)$-memory checker for a data structure $\mathcal{D}_n^I$ is a probabilistic RAM C such that for every execution with at most q user operations, C takes only t_{pre} preprocessing steps, at most t_{post} postprocessing steps and only t_{op} steps to process each user operation. Additionally, C's private memory is bounded above by s bits and the checker uses at most J instances of $\mathcal{D}_n$. A $(t_{pre}, t_{post}, t_{op}, s, q, J)$-memory checker for $\mathcal{D}_n^I$ is called (t, δ, ϵ)-secure if the following holds for every adversary A running in time t:

- *Completeness: If the output of D_n^J is correct for all operations issued by C, then the probability that C returns* BUGGY *or that not all answers of C for the user operations are correct is at most δ, where the probability is taken over the coin tosses of C and A.*
- *Soundness: If the output of D_n^J is false for some operation, then C should output* BUGGY *with probability at least $1 - \epsilon$.*

In most settings we are interested in checkers for which $\delta = 0$ holds. These checkers are called *complete*. Definition 4.1 doesn't rule out the trivial solution, that C simply keeps all values in his private memory. This would rather prevent errors and guarantee correct outputs than check the data structure. We are interested in checkers using only a few bits private memory and causing a small overhead.[3] So this trivial solution gives us an upper bound and a starting point to build more efficient solutions. A checker is called an *on-line checker* iff it outputs BUGGY in that round in which an error occurs. Otherwise it is called an *off-line checker*. A checker is called *noninvasive* if at the end of each round, the insecure memory contains only values specified by the input operations when the checker reads the next operation. Otherwise it is called *invasive*. In particular, our checker based on IncXMACC is off-line and noninvasive with the additional property that the checker passes only user operations to the implementation.

4.2 Designing Checkers via Incremental Schemes

In this section we show how we can derive a memory checker from IncXMACC. We prove that we can check any data structure based on the structure List_n, where List_n represents a list with elements from $\{0,1\}^n$. The initial configuration is empty. List_n supports four operations: $\mathsf{insert}(i, v)$ to insert element $v \in \{0,1\}^n$ at position i, $\mathsf{delete}(i)$ to remove the element at position i and return this value

[3] Note that we don't charge the checker's running time e.g. for inserting or deleting an element using insert and delete commands passed to the implementation (except for the time to write the operation and to read the answer).

to the user, $\mathsf{replace}(i, v)$ to replace the i^{th} value by v and return this element, and $\mathsf{read}(i)$ to return the i^{th} element to the user.

We can design checkers for other data structures based on List_n like stacks and queues. If the checker maintains a counter for the number m of elements currently in the list, the stack resp. queue commands pop, $\mathsf{push}(v)$, $\mathsf{dequeue}$ and $\mathsf{enqueue}(v)$ are equivalent to $\mathsf{delete}(m)$, $\mathsf{insert}(m+1, v)$, $\mathsf{delete}(1)$ and $\mathsf{insert}(m+1, v)$. If the data structure can be implemented with lists, we can combine the checker's program and the list implementation of the data structure to obtain a method to securely store the data of this structure on an insecure medium. The following notion of a sound scheme will help us to prove stronger security:

Definition 4.2 (Sound $\mathcal{M}$-incremental Scheme)
Let $\mathcal{S}(b, s) = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{IncSig}, \mathsf{Vf})$ be an $\mathcal{M}$-incremental authentication or signature scheme. $\mathcal{S}(b, s)$ is called sound iff for all keys produced with positive probability by Gen the following holds: Let M be a message that is obtained by applying a text modification $\pi \in \mathcal{M}$ with argument y to documents $M_1, \ldots, M_m$ and let $\mu_1, \ldots, \mu_m$ and $\mu = \mathsf{IncSig}(M_1, \ldots, M_m, \mu_1, \ldots, \mu_m, \pi, y)$ the corresponding (valid or invalid) signatures. If $\mathsf{Vf}(M, \mu) = 1$, then $\mathsf{Vf}(M_i, \mu_i) = 1$ holds for all $i = 1, \ldots, m$.

Informally, a sound scheme is a scheme such that applying IncSig with an invalid signature μ_i for some M_i doesn't yield a valid signature for M. Note that the soundness property doesn't guarantee security. It only states that one cannot produce a valid signature form invalid signatures *directly*. It may yet be possible to deduce a valid signature from an invalid one.

Lemma 4.3
The $\{\mathsf{delete}, \mathsf{insert}\}$-incremental scheme $\mathsf{IncXMACC}_{F,b}$ is sound.

The proof is omitted. One can easily verify that the tree scheme is sound, too.

Theorem 4.4
Let F be a function family with input length l, output length L and key length κ. Assume that $\mathsf{IncXMACC}_{F,b}$ is $(t, \vec{q}, \vec{L}, \epsilon)$-secure against message substitution attacks for block size $b = n$. Then there exists a non-invasive $(t_{\text{pre}}, t_{\text{post}}, t_{\text{op}}, s, q, I)$-off-line checker for List_n^I, which is $(t', 0, \epsilon)$-secure where

$$t_{\text{pre}} = \text{Time}(\mathsf{FGen}), \quad t_{\text{post}} = c_1 q \cdot \text{Time}(F), \quad t_{\text{op}} = c_1 \cdot (\text{Time}(F) + \log q),$$
$$s = c_2 \cdot (n + l + q \log q + IL + \text{Space}(F)) + \kappa,$$
$$t' = t - c_3(q t_{\text{op}} + t_{\text{pre}} + t_{\text{post}}), \quad q_i = q, \quad I = \min\{q_s, q_v\}.$$

for small constants $c_1, c_2, c_3 \in \mathbb{N}$. Here, $\text{Time}(F)$ resp. $\text{Space}(F)$ denotes the time resp. space to evaluate a function from F and $\text{Time}(\mathsf{FGen})$ denotes the time to draw a key for a function in F.

A sketch of the proof is given in Appendix B. It is easy to see that we can derive an on-line checker for List_n from the tree scheme. Storing the signature in the checker's private memory is too expensive. Hence, we need additional instances to store the nodes of the signature tree on the insecure memory. In this case, security is provided by the fact that the tree scheme is secure against total substitution attacks. However, this checker is invasive and we cannot for example efficiently apply this construction to stacks, because in this case we cannot access all parts of the signature fast.

4.3 A Lower Bound for Substitution Detecting Schemes

First, we define a *normal form* for adversaries performing attacks on the message substitution detection property. Let $\mathcal{S}(b,s) = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{IncSig}, \mathsf{Vf})$ an $\mathcal{M}$-incremental (signature or authentication) scheme. We assume that IncSig outputs the invalid signature $\perp$ if, for some reason, it refuses to produce a valid one. An attack on the detection property is a message substitution attack, such that each IncSig query $(\alpha_1, \ldots, \alpha_m, \beta, \pi, y)$ has the following form:

1. The adversary may replace any message M_{α_i} with $M^*_{\alpha_i}$ by alter commands. Let $M^*_{\alpha_i}$, $i = 1, \ldots, m$, be this sequence of messages (where we allow $M^*_{\alpha_i} = M_{\alpha_i}$). Additionally, the adversary stores the current content M_β.
2. The adversary queries IncSig for $(\alpha_1, \ldots, \alpha_m, \beta, \pi, y)$.
3. The adversary replaces all messages with name α_i by M_{α_i} again. If IncSig has returned $\perp$, the adversary replaces the document with name β by the former value.

Furthermore, the adversary doesn't use additional alter commands. It is easy to see that every adversary can be assumed w.l.o.g. to be in normal form. Therefore, we can associate each alter command uniquely to an IncSig query. If IncSig doesn't return $\perp$ in step 2, the adversary may either replace M_β again or not.

For notational convenience, let $M[i] = \star$ for the message $M[1] \cdots M[n]$ and $i > n$, where $\star$ denotes a special symbol $\star \notin \Sigma$. In particular, we have $M[i] \neq M'[i]$ for messages $M[1] \cdots M[n]$ and $M'[1] \cdots M'[n']$ with $n < i \leq n'$.

Definition 4.5 (Successful Adversary)
A (normal form) adversary for the detection property is successful, if IncSig *returns in step 2 a signature different from $\perp$ for a query $(\alpha_1, \ldots, \alpha_m, \beta, \pi, y)$, such that for the blocks $M^*_{\alpha_{i_h}}[j_h]$, $h = 1, \ldots, k$, that* IncSig *has read to produce this signature, we have $M^*_{\alpha_{i_h}}[j_h] \neq M_{\alpha_{i_h}}[j_h]$ for some $h \in \{1, \ldots, k\}$.*

Note that Definition 4.5 doesn't rule out the trivial solution that IncSig always outputs $\perp$ resp. that IncSig never reads a block.

Definition 4.6 (Message Substitution Detecting Scheme)
Let $\mathcal{S}(b,s) = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{IncSig}, \mathsf{Vf})$ be an $\mathcal{M}$-incremental scheme. A $(t, \vec{q}, \vec{L}, \delta)$-adversary for the detection property is specified by the parameters in definition 2.2, where δ is the success probability. $\mathcal{S}(b,s)$ is called $(t, \vec{q}, \vec{L}, \delta)$-detecting, if there exists no $(t, \vec{q}, \vec{L}, \delta)$-adversary for the detection property.

Thus, message substitution detecting schemes can be viewed as on-line checkers. To prove that a detecting scheme which is secure against basic attacks, is also secure against message substitution attacks, we need the following definition:

Definition 4.7 (Scheme with Predictable IncSig-Access)
The $\mathcal{M}$-incremental scheme $\mathcal{S}(b,s) = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{IncSig}, \mathsf{Vf})$ is a scheme with p-predictable IncSig*-access, iff one can for all (with positive probability generated) keys, all messages M_{α_i} with $M_{\alpha_i} = M_i[1] \cdots M_i[n_i]$ and signatures μ_{α_i}, $i = 1, \ldots, m$, predict the message blocks, which* IncSig *accesses*

to update the signature in response to $(\alpha_1, \ldots, \alpha_m, \beta, \pi, y)$ *in time* $p(\max\{n_i\})$ *(in the corresponding computational model) from* μ_{α_i}, $i = 1, \ldots, m$, *and* π, y.

For simplicity, we have assumed that IncSig's access is predictable from μ_{α_i}, π, y in time $p(\max\{n_i\})$. Extensions to other parameters are straight forward. Clearly, the tree scheme is a message substitution detecting scheme with predictable IncSig-access.

Proposition 4.8
Let $\mathcal{S}(b,s) = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{IncSig}, \mathsf{Vf})$ *be a* $(t, \vec{q}, \vec{L}, \delta)$*-detecting* $\mathcal{M}$*-incremental scheme with p-predictable* IncSig*-access, which is* $(t, \vec{q}, \vec{L}, \epsilon)$*-secure against basic attacks. Then* $\mathcal{S}(b,s)$ *is* $(t', \vec{q}, \vec{L}, \epsilon')$*-secure against message substitution attacks, where* $t' = t - q_i p(L_i)$ *and* $\epsilon' = \epsilon + \delta$.

Proof (Sketch). Let E be a normal form adversary with parameters $t, \vec{q}, \vec{L}$, which is successful with probability at least ϵ in a message substitution attack. From E we construct via black-box-simulation an adversary A performing a basic attack.

A simulates each query E to Sig and Vf by its oracle access to $\mathcal{S}(b,s)$. If E issues an IncSig query without having used an associated alter command in step 1 of the normal form specification, then A passes this query to IncSig and returns the signature to E. Assume, that E tampers messages M_{α_i} to $M^*_{\alpha_i}$ before. Then A computes in time $p(L_i)$ from μ_{α_i}, $i = 1, \ldots, m$, and π, y the message blocks $M^*_{\alpha_{i_h}}[j_h]$, $h = 1, \ldots, k$, which IncSig would read. If $M^*_{\alpha_{i_h}}[j_h] \neq M_{\alpha_{i_h}}[j_h]$ for some h, A returns $\perp$ to E without quering IncSig. Else A passes the query to IncSig without tampering the messages and returns the signature to E. In this case, the signature does not depend on other (altered or unaltered) blocks and the answer is correct.

As alter commands don't change virtual documents, every virtual document appearing in A's attack appears in E's attack as well. Let Detect be the event, that E isn't successful in an attack for the detection property. Furthermore, let Succ_A resp. Succ_E be the events that A resp. E performs a successful attack on the signature scheme. We have

$$\epsilon' \leq \mathrm{Prob}[\mathsf{Succ}_E] \leq \mathrm{Prob}[\mathsf{Succ}_E \mid \mathsf{Detect}] + \mathrm{Prob}[\neg\ \mathsf{Detect}] \leq \mathrm{Prob}[\mathsf{Succ}_A] + \delta.$$

Hence, A is successful with probability at least ϵ. ■

The following proposition shows that we cannot design detecting schemes producing small signatures:

Proposition 4.9
Let $\mathcal{S}(b,s)$ *be a complete* $(t, \vec{q}, \vec{L}, \delta)$*-detecting incremental scheme for* $t = cbn$, $q_s = 1$, $q_i = n$, $L_s = L_i = n$, *which supports the* replace *modification such that* IncSig *always accesses the* i^{th} *block for valid* $\mathsf{replace}(M_\alpha, i, M_*)$ *commands. Then for* $\Delta := 1 - \delta > \frac{1}{2}$ *the bit length of a signature for a message* $M = M[1] \cdots M[n]$ *must be at least*

$$(1 - \beta)\tfrac{n}{t_{\max}} + \log_2 \gamma,$$

where $\beta = 1 - 2(\alpha - \frac{1}{2})^2 \log_2 e < 1$, $\gamma = \frac{\Delta - \alpha}{1 - \alpha} < 1$ *for* $\frac{1}{2} < \alpha < \Delta$. *Here,* $t_{\max}$ *is the maximal number of blocks* IncSig *reads for an update step.*

The proof is omitted. If Δ and α are close to 1, we have $1 - \beta \approx \frac{1}{3}$ and $\gamma \approx 1$, i.e. a signature must have at least $\frac{n}{3t_{\max}}$ bits. Note that this bound holds for all $n, t_{\max}$ and δ.

Acknowledgements

We thank Roger Fischlin for pointing out the topic of memory checkers and C.P. Schnorr and the anonymous referees for their comments. We also thank Mihir Bellare and Daniele Micciancio for discussions about their works.

References

[AHU74] A.Aho, J.Hopcroft, J.Ullman: The Design and Analysis of Computer Algorithms, *Addison Wesley*, 1974.

[BGG94] M.Bellare, O.Goldreich, S.Goldwasser: Incremental Cryptography: The Case of Hashing and Signing, *Crypto '94, Lecture Notes in Computer Science, Vol. 839, Springer-Verlag, pp.* 216–233, 1994.

[BGG95] M.Bellare, O.Goldreich, S.Goldwasser: Incremental Cryptography and Application to Virus Protection, *Proceedings of the 27th Annual ACM Symposium on the Theory of Computing, pp.* 45–56, 1995.

[BGR95] M.Bellare, R.Guérin, P.Rogaway: XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions, *Crypto '95, Lecture Notes in Computer Science, Vol. 963, Springer-Verlag, pp.* 15–29, *extended version available from* `http://www.cs.ucdavis.edu/~rogaway/`, 1995.

[BKR94] M.Bellare, J.Killian, P.Rogaway: On the Security of Cipher Block Chaining, *Crypto '94, Lecture Notes in Computer Science, Vol. 839, pp.* 341–358, 1994.

[BEG+94] M.Blum, W.Evans, P.Gemmell, S.Kannan, M.Naor: Checking the Correctness of Memories, *Algorithmica, Volume 12, pp.* 225–244, 1994.

[BK89] M.Blum, S.Kannan: Designing Programs that Check Their Work, *Proceedings of the 21st Annual ACM Symposium on the Theory of Computing, pp.* 86–97, 1989.

[GGM86] O.Goldreich, S.Goldwasser, S.Micali: How to Construct Random Funtions, *Journal of ACM, Vol. 33(4), pp.* 792–807, 1986.

[GMR89] S.Goldwasser, S.Micali, C.Rackoff: The Knowledge Complexity of Interactive Proof Systems, *SIAM Journal on Computation, Vol. 18, pp.* 186–208, 1989.

[GMR88] S.Goldwasser, S.Micali, R.L.Rivest: A Digital Signature Scheme Secure Against Adaptive Chosen Message Attacks, *SIAM Journal on Computation, Vol. 17(2), pp.* 281–308, 1988.

[GO96] O.Goldreich, R.Ostrovsky: Software Protection and Simulation on Oblivious RAM, *Journal of ACM, Vol. 43(3), pp.* 431–473, 1996.

[M97] D.Micciancio: Oblivious Data Structures: Application to Cryptography, *(to appear at) Proceedings of the 29th Annual Symposium on the Theory of Computing*, 1997.

A Sketch of Proof of Theorem 3.1

We use Theorem 6.1 from [BGR95]. We prove that breaking the incremental scheme in a message substitution attack yields a successful attack for a suitable, but "artificial" non-incremental scheme. First, we describe a non-incremental scheme that allows to produce MAC. Then we show that we can simulate the adversary E for the incremental scheme to obtain an adversary A for the non-incremental scheme. Finally, we show an upper bound for the probability breaking the non-incremental scheme.

Let the parameters of the non-incremental XOR scheme be $b' = 2b$, $l' = l$ and $L' = L$. The scheme is specified by the algorithms $\mathcal{R}$ and $\mathcal{E}$. The $\mathcal{R}$ algorithm holds two counters dcnt and bcnt and a bit s which are all initialized with 0. The bit s specifies how messages with less than three blocks are dealt with. On input a message $M = M[1] \cdots M[n]$ algorithm $\mathcal{R}$ proceeds as follows: Let $M_L[i]$ ($M_R[i]$) be the left (right) half of $M[i]$ and $n \geq 3$. Then $\mathcal{R}$ verifies that $M_R[1] = \text{bcnt}$ and that $M_R[i+1] = M_R[i] + 1$ for $i = 1, \ldots, n-1$. If not, it gives an undefined output $\perp$ and the signer refuses to give a signature. Otherwise $\mathcal{R}$ increments dcnt, outputs this value and overwrites the last saved message with M in his history. Furthermore, it sets $s = 0$ and increments bcnt by n.

The cases $n = 1$ and $n = 2$ are used to simulate deletion and insertion if some message substitution has occured before. Assume that the queried message M has only one block and let $H = H[1] \cdots H[m]$ be the message stored in the history. $\mathcal{R}$ searches for the uniquely determined $i \in \{1, \ldots, m\}$ with $M_R[1] = H_R[i]$ and $M_L[1] \neq H_L[i]$. If no such i exists or $m \leq 3$, $\mathcal{R}$ outputs $\perp$ and the signer refuses to give a signature. Otherwise $\mathcal{R}$ increments dcnt and outputs $(\text{dcnt}, i, m, H[i-1], H[i], H[i+1])$. (If $i = 1$ or $i = m$, output $0^{b'}$ instead of $H[i-1]$ or $H[i+1]$). Furthermore, it deletes $H[i]$ from the history and sets $s = 1$ to indicate that some substitution has occured. We skip the case $n = 2$.

We define the output of $\mathcal{E}$. Let $l^* = \frac{1}{2}l - 1$. The input is some seed r produced by $\mathcal{R}$ and a message $M[1] \cdots M[n]$. If $n \geq 3$, then r equals dcnt, so $\mathcal{E}$ produces the following set $Z \subseteq \{0,1\}^l$:

$$Z = \{0 \cdot r\} \cup \left\{10 \cdot \langle M[i] \rangle_{l-2} \;\middle|\; i = 1, \ldots, n\right\} \cup \{11 \cdot \langle M_R[i] \rangle_{l^*} \cdot \langle M_R[i+1] \rangle_{l^*} \mid i = 1, \ldots, n-1\}$$

Note that this is equivalent to the output of the Sig algorithm in our incremental scheme as we've checked that the right halves of the message blocks are in increasing order starting with the previous value of bcnt.

Let $n = 1$. Then r has the form $(d, i, m, H[i-1], H[i], H[i+1])$ and $\mathcal{E}$ outputs

$$\begin{aligned} Z = &\{0 \cdot \langle d-1 \rangle_{l-1}\} \cup \{0 \cdot \langle d \rangle_{l-1}\} \cup \{10 \cdot \langle H[i] \rangle_{l-2}\} \cup \{11 \cdot \langle H_R[i-1] \rangle_{l^*} \cdot \langle H_R[i] \rangle_{l^*}\} \\ &\cup \{11 \cdot \langle H_R[i] \rangle_{l^*} \cdot \langle H_R[i+1] \rangle_{l^*}\} \cup \{11 \cdot \langle H_R[i-1] \rangle_{l^*} \cdot \langle H_R[i+1] \rangle_{l^*}\} \end{aligned}$$

If $i = 1$ or $i = m$, which can be detected by $\mathcal{E}$, change the last line accordingly. Again, we skip the case $n = 2$.

We describe how we obtain a black-box-simulation of the adversary E for the incremental scheme. The adversary A for the non-incremental scheme works as follows: It holds a counter bcnt$'$ initialized with 0 to mimic the block counter of the non-incremenral $\mathcal{R}$ algorithm and a bit s' initialized with 0 to store whether E has altered the current document or not. Every time E issues a create command for $M = M[1] \cdots M[n] \in \{0,1\}^{bn}$, $n \geq 3$, A passes a signing query to the non-incremental scheme for $M' = M'[1] \cdots M'[n] \in \{0,1\}^{b'n}$ with $M'[i] = M[i] \cdot \langle \text{bcnt}' + i \rangle_b$. Then A augments the MAC (d, z) with the values $\text{bcnt}'+1, \ldots, \text{bcnt}'+n$ and returns it to E. This extended MAC equals the MAC produced by the incremental Sig algorithm. A increments bcnt$'$ by n, stores M' in his memory for further use and sets $s' = 0$.

W.l.o.g. we assume that E issues alter commands only before a delete(i) command and that this alter command substitutes the i^{th} block by a different block, because message substitutions for an insert command or any other block don't affect the new MAC. Hence, the alter commands can be thought of having the form alter(M_*, i) for $M_* \in \{0,1\}^b$. If E uses this comand, A replaces block $M'[i]$ by $M_* \cdot M'_R[i]$ and sets $s' = 1$ to indicate that some alternation has occured.

Let E issue a delete(i) command for the current document $M' = M'[1] \cdots M'[n]$. If $s' = 0$, i.e. $M[i]$ hasn't been altered, we delete the i^{th} block in the stored message M' and query the non-incremental scheme for this new message M'. Again, we augment the MAC (d, z) by $M'_R[1], \ldots, M'_R[n-1]$ and return it to E. If $s' = 1$, i.e. some block has been altered previously, A passes the single block message $M'[i]$ to the non-incremental signer and receives a MAC having the form $(d, i, n, M'[i-1], M'[i], M'[i+1], z)$ and returns $(d, M'_R[1], \ldots, M'_R[i-1], M'_R[i+1], \ldots, M'_R[n], z')$ to E, where $z' = \bigoplus_{j=1}^{k} z_j$ and $(d_1, z_1), \ldots, (d_k, z_k) = (d, z)$ are the previously returned MACs, where (d_1, z_1) is the last MAC having received with $s' = 0$. Note that the MAC, which E receives, equals the MAC that would have been returned by the incremental scheme. A deletes $M'[i]$ from M'. Again, we skip the case insert(i, M_*).

One can easily show that if E outputs a document M with more than two blocks which hasn't appeared as a virtual document before, then M hasn't been signed in A's attack. The proof that the success probability for an adversary breaking the non-incremental scheme is bounded above by δ_I is omitted.

B Sketch of Proof of Theorem 4.4

Clearly, the checker runs the incremental scheme IncXMACC to check the correctness. For every instance we'll have a signature for the content. Updating this signature when inserting, deleting, replacing or reading an element will be done with the insert, delete commands for the incremental scheme. To prevent repetition attacks, we prepend every "message" with a time stamp which the checker stores in its local memory, not in the insecure memory. This time stamp is updated before processing insert, delete commands.

If no more operations are left, the checker empties the memory in a postprocessing phase: For each initialized instance it deletes the values in the instance using delete commands and checks that the obtained signatures are accepted by Vf. If some signature is not accepted, it outputs BUGGY, otherwise C accepts.

If all operations work correctly, the checker never outputs BUGGY since IncXMACC is complete. Assume that there is a sequence of operations such that the checker is fooled. We design a adversary E for IncXMACC. E works as follows: Let A be the adversary for the checker. Then E first runs the whole execution simulating C and A by black-box-simulation using the oracle access for the incremental scheme. Moreover, E maintains the correct memory contents and stores all signatures.

Since E has simulated the whole execution first, he knows the last user operation for which a wrong value has been returned. E builds a message M that consists of the time stamp, the correct memory content (at this point) and replaces the corresponding block with the wrong value. E outputs this message M and the signature μ for this message as a forgery. As the scheme is sound and the checker doesn't output BUGGY, i.e. the signature for the final value has been accepted, this signature μ is valid for M. Virtual documents are only changed by insert and delete commands, therefore all virtual documents are defined by the correct memory content and the counter values. Since there is some error in M, and the time stamps make every virtual document unique, M hasn't appeared as a virtual document during the execution. Hence, E is successful whenever A is.