Parallelizable Authentication Trees

Eric Hall and Charanjit S. Jutla
IBM T.J. Watson Research Center,
Yorktown Heights, NY 10598-704

Abstract

We define a new authentication tree in the symmetric key setting, which has the same computational time, storage and security parameters as the well known Merkle authentication tree, but which unlike the latter, allows for all the cryptographic operations required for an update to be performed in parallel. The cryptographic operations required for verification can also be parallelized. In particular, we show a provably secure scheme for incremental MAC with partial authentication secure against substitution and replay attacks, which on total data of size 2^n blocks, and given n cryptographic engines, can compute incremental macs and perform individual block authentication with a critical path of only one cryptographic operation

1. Introduction

We design a novel incremental MAC (message authentication code) with partial authentication secure against substitution and replay attacks. Before we give detailed definitions, to motivate the defintions and as an application consider the following problem of checking correctness of memory [5].

In this application, a tamper proof or secure processor is using an insecure storage device (e.g. RAM), open to attack from an adversary who can read and modify the RAM. However, the secure processor may also have a small amount of internal memory which can store a MAC of the whole unprotected RAM. The secure processor is required to generate incremental MACs and authenticate individual blocks of RAM (without computing the MAC on the whole RAM). Of course, these computations should not substantially deteriorate the performance of the overall system. Similar situation arises in NAS (network attached storage) systems.

Clearly, there are two extreme (though impractical) solutions to this problem. One is to store the whole RAM (say 2^n blocks) inside the secure device. The other is to have 2^n cryptographic engines (e.g. AES or SHA-1) inside the secure device which can compute/authenticate a MAC of the whole unprotected memory using a parallel MAC scheme like XOR-MAC [1], with a critical path of one cryptographic operation. We stress here that although XOR-MAC can compute incremental

MACs with only one engine, to verify an individual block of RAM, it must compute the MAC on the whole RAM (i.e. XOR-MAC is *not* incremental with respect to verification). The ability to verify only a single block (without having to compute the whole MAC) is a crucial requirement of our problem.

One could try a memory/processor tradeoff, by dividing the unprotected memory into superblocks (say of size 2^m blocks each), and storing an XOR-MAC (one block) of each superblock inside the secure device, and computing/authenticating an XOR-MAC in parallel using 2^m cryptographic engines. Now the memory required to store the MACs has been reduced to 2^{n-m} blocks[1]. Note however, that the number of engines times the secure memory required remains 2^n.

The main contribution of this paper is a provably secure scheme for this problem (incremental MAC with partial authentication secure against substitution and replay attacks), which with only n cryptographic engines, and 1 block of secure memory, can compute incremental macs (and do individual block authentication) with a critical path of one cryptographic operation. The *only* overhead is an increase in size of the unprotected memory by a factor of two.

Before we describe our scheme, lets describe the other existing solution to this problem. This solution of [5] uses Merkle authentication trees [14]. However, Merkle trees are not parallelizable (i.e. although a Merkle tree based solution would only require n cryptographic engines, and 1 block of secure memory, the critical path of an incremental MAC computation would be n cryptographic operations). Not surprisingly though, as we will soon see, ours is also a tree based solution.

Main Result As for XOR-MAC, for every finite secure PRF (pseudorandom function) family F, we construct an *incremental MAC with partial authentication secure against substitution and replay attacks.* The key difference from XOR-MAC is that our scheme does efficient parallel partial authentication. To acheive our goal, the scheme we describe generates auxiliary data which can be stored in unprotected memory. In other words, when provided with correct auxiliary data a single block can be authenticated, whereas no adversary can get a wrong block authenticated even with auxiliary data of its choice.

Surprisingly, the MAC is just a random number (or a counter value) chosen independently of the data to be authenticated! Its the auxiliary data which provides complete authentication, and in a way we are trying to make the naive solution of "storing MACs outside" work (see footnote). Informally, the scheme works as follows. Each pair of data blocks is locally MACed along with a new random value (or counter), and each pair of these random values (or counters) are locally MACed along with yet another random value (or counter) at a higher level, and so on, till we reach the root of this tree. The random value (or counter) at the root is the MAC of the scheme. The

[1] Note that storing MACs outside in unprotected memory only provides integrity and not protection against replay attacks

local MACs are all stored outside in unprotected memory as auxiliary data. We stress that we are not locally MACing two children nodes with the value at the parent as key, but rather MACing all three values together using a global secret key. The former would lead to an insecure solution. The latter can be seen as a tweakable MAC (cf. tweakable block ciphers [12]), i.e. we use the parent value as a tweak.

Note that all the local computations can be done in parallel once the random numbers (or counters) have been chosen. Efficient incrementality follows as an update in a data block only requires updates on the path from this leaf to the root (and the sibling path – which we will describe later). Efficient parallel partial authentication, i.e. authentication of data at a leaf, follows as it requires checking the local MACs on the path from this leaf to the root. This latter crucial property of partial authentication is what makes the scheme different from XOR-MAC.

The only thing that remains to be seen is that we do not reuse counters, and that the adversary cannot move around the auxiliary data, which is the crux of the proof of security.

Various optimizations result from how the local MACs are computed. As mentioned earlier, our scheme while providing the additional property of partial authentication over XOR-MAC, has a space overhead in unprotected memory of a factor of two (i.e. same overhead as required by Merkle hash tree schemes). All these issues are addressed in section 5.

We finally describe how an XOR-MAC like scheme PMAC(which uses XOR universal hash function families) can be extended to provide efficient partial authentication.

The appendix contains an earlier version of the paper (which some readers prefer) posted on the iacr archive on Dec 12, 2002. It also describes an incremental authenticated encryption scheme.

2. Definitions

As is often conveniently done, for a function F with many parameters, if the first parameter is a key then F_x will denote the function with the first parameter x. The symbol $||$ will denote concatenation of strings. For a message M consisting of m blocks $M_1, ..., M_m$, $M\langle i, a\rangle$ will denote the modified message with the ith block M_i replaced by a.

We first give the definition of simple MAC and an adversary's success probability. Although a MAC scheme is usually defined as a triplet of key generation, mac, and verify algorithms, the following definition suffices. For more rigorous definitions see [2].

Definition 2.1 A simple MAC scheme consists of a function F which takes a secret key of m bits, a plaintext of k bits and produces a value of e bits.

Security of the simple MAC scheme is defined using the following experiment. An oracle adversary A is given access to oracle $F_x(\cdot)$, where x is chosen uniformly at random. After A queries

$F_x(\cdot)$ on plaintexts $M_1, M_2, \ldots M_l$ (adaptively), it outputs a pair M', τ', where M' is not one of the queried plaintexts. The adversary's success probability is given by

$$\Pr[\mathrm{F}_x(M') = \tau']$$

Let Sec-MAC$_F(l, t)$ be the maximum success probability of any adversary running in time at most t, and making at most l queries.

Definition 2.2 An *incremental MAC with partial authentication and with auxiliary data* (**IMACAUX**) consists of the following:

- MAC-AUX: MAC-AUX is a probabilistic function with arguments a key x of m bits, plaintext M of size at most 2^n blocks, each block of size d bits, and produces a tuple $< \sigma, \tau >$, where σ can be an arbitrarily long string, and τ is of size e bits. The string σ will be called auxiliary data and τ will be called authentication tag.

  We will write [MAC-AUX$_x(M)$] for all tuples $< \sigma, \tau >$ which have non-zero probability of occurring as MAC-AUX$_x(M)$, i.e. with non-zero support.

- Verify: Verify is a boolean function which takes a key x of m bits, an index $i \in [0..2^n - 1]$, a d bit block a, σ and τ, with the following property: Verify$_x(i, a, \sigma, \tau) = 1$ if there exists an M, such that the ith block of M is a, and $< \sigma, \tau >$ is in the support of MAC-AUX$_x(M)$.

- INC-MAC-AUX (update): INC-MAC-AUX is a probabilistic function which takes (apart from the key x) an index i, a block of plaintext a, σ, and τ, and produces either a tuple $< \sigma', \tau' >$ or $\perp$. If there exists an M such that $< \sigma, \tau >$ is in the support of MAC-AUX$_x(M)$ then it must return a $< \sigma', \tau' >$ such that $< \sigma', \tau' >$ is in the support of MAC-AUX$_x(M\langle i, a\rangle)$.

We have said nothing about the security of IMACAUX, which we address next.

Definition 2.3 (*Security under substitution and replay attacks*) The security of an IMACAUX scheme $\langle$ MAC-AUX, Verify, INC-MAC-AUX$\rangle$ is defined using the following experiment. A three oracle adversary A is given access to oracles MAC-AUX$_x(\cdot)$, $Verify_x(\cdot,\cdot,\cdot,\cdot)$, and INC-MAC-AUX$_x(\cdot,\cdot,\cdot,\cdot)$, where x is chosen uniformly at random from $\{0,1\}^m$. The adversary first requests an initial MAC-AUX$_x$ to be computed on an initial plaintext M^0 of 2^n blocks. Let MAC-AUX$_x(M^0)$ return $< \sigma_0, \tau_0 >$.

Subsequently, adversary requests a sequence of q adaptive update operations, each specifying a block number and a block of replacement text, along with auxiliary data of its choice. However, τ supplied on each request must be same as that produced in the previous request.

More precisely, for each i, $q \geq i > 0$, let j_i be the block number for the ith incremental update, with text a_i. Let $M^i = M^{i-1}\langle j_i, a_i\rangle$. Let $I_i = \text{INC-MAC-AUX}_x(j_i, a_i, \sigma'_{i-1}, \tau_{i-1})$, where j_i, a_i, and σ'_{i-1} are adaptively chosen by the adversary. The return value I_i is either $\perp$ or $< \sigma_i, \tau_i >$.

Finally, the adversary requests a verification on a block at index j, with text a, such that a is different from M^q_j. The adversary's *success probability* is given by

$$\Pr[(\forall i \in [1..q] \ : I_i \neq \perp) \wedge \text{Verify}_x(j, a, \sigma', \tau_q) = 1]$$

Again, σ' is adaptively chosen by the adversary. However, τ_q remains the same as produced by the last update. We stress that a is *only required to be different from the last plaintext* at block j, and the adversary is allowed to choose an a at index j, along with a σ', which may have occurred at an earlier point of time.

Let $\text{Sec-IMACAUX}_{\langle \text{MAC-AUX,Verify, INC-MAC-AUX} \rangle}(q, t)$ be the maximum success probability of any adversary running in time at most t, and making at most q INC-MAC-AUX queries.

3. Parallelizable Authentication Tree

We now describe an IMACAUX scheme called PAT with a description of each of its component functions, i.e. MAC-AUX, Verify and INC-MAC-AUX. The functions will employ a simple MAC F (see definition 2.1) with the same secret key x as chosen for PAT. We will describe the various size parameters later (before Theorem 1). All F computations (which will be the only cryptographic operations) in the computation of these functions can be done in parallel.

MAC-AUX

Given a 2^n block plaintext M, we now describe a valid MAC-AUX on it, i.e. all pairs $< \sigma, \tau >$ which are in the support of $\text{MAC-AUX}_x(M)$. $\text{MAC-AUX}_x(M)$ will be a k-ary labelled tree (see Figure 1). For simplicity, we only consider k=2. The tree will be balanced and will have 2^n leaves.

- the labels at each internal non-root node u will be a nonce value $V(u)$, and a local mac value $C(u)$ such that $C(u) = \text{F}_x(V(u)||V(\text{parent}(u))||\lambda)$, where $||$ is the concatenation operator, and λ is a bit which is 0 is u is the left child, and 1 is it is the right child (see Fig 1). Note that this is slightly different form what was described in the introduction, but this version is easier to understand.
- The root node r will only have as label a nonce value $V(r)$, which will constitute the authentication tag τ.
- Leaf nodes u will have as label a data value $data(u)$ which will be the corresponding block

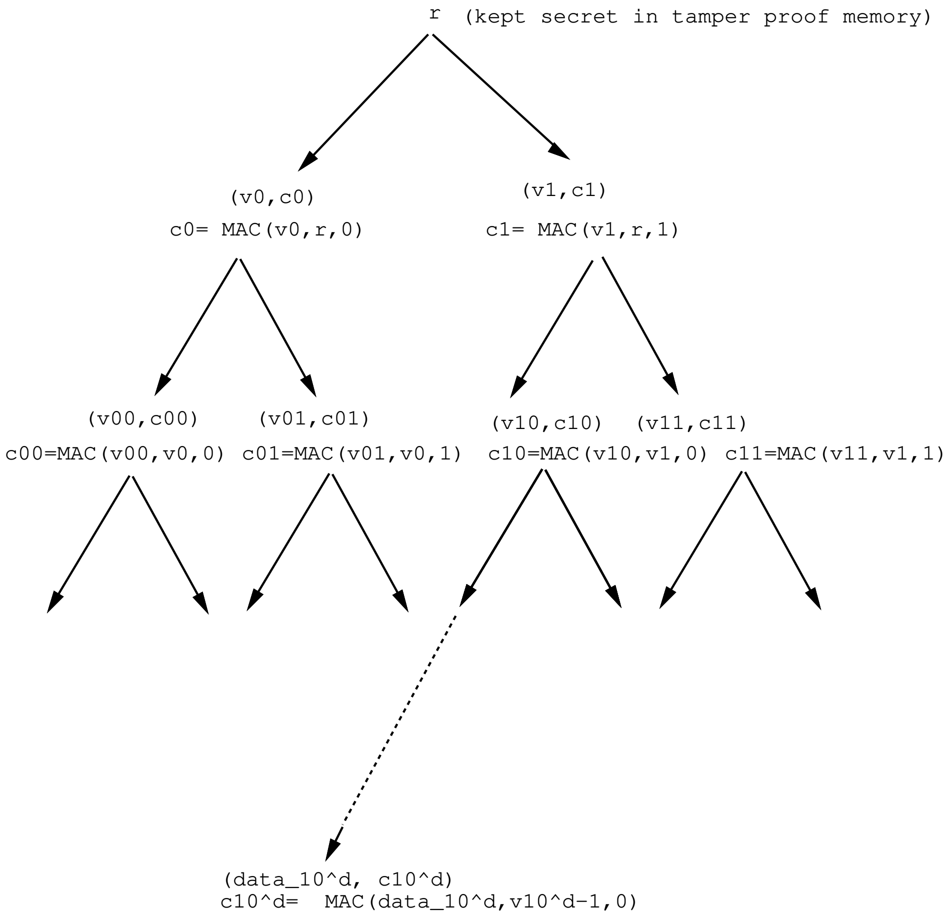

Figure 1: Two Levels of an Authentication Tree with a Leaf

from M, and a mac label $C(u)$ such that $C(u) = \mathrm{F}_x(data(u)||V(\mathrm{parent}(u))||\lambda)$, where λ is as before.

Thus, σ is just the whole labelled tree except the root label. Again, τ is the V label of the root node. Any such $< \sigma, \tau >$, which satisfy the local F constraints, is in the support of MAC-AUX(M).

Verify

Let the input to the function be (i, a, σ, τ).

Since the boolean function Verify takes the leaf node index as an argument, it just checks that the C values are consistent with F_x along the path from this leaf to the root. We will give an algorithmic description of this function.

More precisely, let the path from the root to the specified leaf z (the ith leaf) be $u_0 = r$, $u_1, ..., u_{n-1}$, $u_n = z$. Recall that $V(u_0)$ is τ. In the following, λ will be 0 if u_y is a left child, and 1 otherwise.

If for all $y = 1$ to $n - 1$,
  $C(u_y)$ equals $\mathrm{F}_x(V(u_y)||V(u_{y-1})||\lambda)$, and
  $C(u_n)$ equals $\mathrm{F}_x(a||V(u_{n-1})||\lambda)$
then return 1, else return 0.

INC-MAC-AUX (update)

Let the input to the function be (i, a, σ, τ). We will give an algorithmic description of this function.

Since the tree has 2^n leafs, the root can be considered to be at the 0th **level**, whereas the leafs are at the nth level. A path from root r to leaf z (the ith leaf) can then be written as $u_0 = r$, $u_1, ..., u_{n-1}$, $u_n = z$. This will be called the *update path*. Let $v_1, ...v_n$ be the *sibling path*, i.e. for each $y \in [1..n]$, v_y is the sibling of u_y. The update algorithm first checks that the nonce values V provided as part of σ of the sibling path nodes are correct. This requires checking that the values V on the update path are correct as well. More precisely,

Step 1 : If for all $y = 1$ to $n - 1$,
  $C(u_y)$ equals $\mathrm{F}_x(V(u_y)||V(u_{y-1})||\lambda)$, and
  $C(v_y)$ equals $\mathrm{F}_x(V(u_y)||V(u_{y-1})||\lambda)$, and moreover,
  $C(v_n)$ equals $\mathrm{F}_x(data(v_n)||V(u_{n-1})||\lambda)$,
  where λ is as before, then perform the update below, else return $\perp$.

Step 2 : For $y = 0$ to $n - 1$, set $V(u_y) = \tau + n - y$.
  For $y = 1$ to $n - 1$, set $C(u_y) = \mathrm{F}_x(V(u_y)||V(u_{y-1})||\lambda)$, where λ is as before.

Step 3 : At the leaf node $z = u_n$, set $data(u_n) = a$, and set $C(u_n) = \mathrm{F}_x(data(u_n)||V(u_{n-1})||\lambda)$.

Step 4 : The update algorithm recomputes and updates $C(v_y)$ for each y, using its given nonce value, and its parent's newly chosen nonce value. Clearly, the bit λ at each v_y is the opposite of what was used at u_y. More precisely,

for $y = 1$ to $n - 1$
set $C(v_y) = \mathrm{F}_x(V(v_y)||V(u_{y-1})||\lambda)$, and
set $C(v_n) = \mathrm{F}_x(data(v_n)||V(u_{n-1})||\lambda)$

The newly re-labelled tree is returned as σ, τ. Note that all the F operations in steps 1 to 4 combined can be done in parallel.

That finishes the description of INC-MAC-AUX.

4. Security of Parallelizable Authentication Tree

Optimized Initialization of the Authentication Tree

Since, in the definition of security which described the adversarial model (see definition 2.3), the adversary only makes one initial call to MAC-AUX, our scheme (i.e. MAC-AUX) will return a σ_0, with all V values set to zero, and similarly the τ_0 set to zero. This can be seen as initializing the data structure. Moreover, with this simple initialization, all the internal node MAC values C are same, and hence need not be computed multiple times. Further still, we will assume that the data values are also initialized to zero, in which case the MAC values at leafs will also be same.

This does not change the adversarial model, since if the adversary requested a different initial plaintext M^0, our algorithm could return the data structure obtained by simulating several updates.

For example, before the first update request, as mentioned, all nonce values are zero. After the first update request, the nonce value of the node closest to the leaf, i.e. u_{n-1} will be 1, and the nonce value of the node closest to the root, i.e. u_1 will be $n - 1$, and the nonce value of the root will just be n.

As we will see, this assures that in each incremental request, the nonce values are chosen afresh, i.e. are never repeated.

Let l be the number of bits in the nonce labels V above.
Let c be the number of bits in the C label.
Let d be the number of bits in each block of data stored at a leaf.
Let m be the number of bits in the key.
Let 2^n be the number of leaves in the balanced binary tree.
Let F be a function $F : \{0,1\}^m \times \{0,1\}^{\max\{l,d\}+l+1} \rightarrow \{0,1\}^c$

The above three algorithms together describe an **IMACAUX** scheme and will be called $PAT^F(m,n,l,c,d)$ (parallelizable authentication tree) when using F as its local MAC.

Theorem 1: For any positive integers m,n,l,c,d,q,t and any function $F : \{0,1\}^m \times \{0,1\}^{\max\{l,d\}+l+1} \rightarrow \{0,1\}^c$

$$\text{Sec-MAC}_F(2qn,t) \;\geq\; \text{Sec-IMACAUX}_{PAT^F(m,n,l,c,d)}(q,t)$$

Proof: Let A be a three oracle adversary as in the experiment of Theorem 2.3. Let B be an oracle adversary which participates in the experiment of Definition 2.1 (simple MAC), and is given access to oracle $F_x(\cdot)$, with x chosen uniformly at random.

Adversary B will simulate the three oracles for A, i.e. MAC-AUX, Verify, and INC-MAC-AUX of PAT using its own oracle $F_x(\cdot)$. B will then just imitate A. During the simulation, B will make several oracle calls to $F_x(\cdot)$. It will also maintain a List of pairs. Let List_j denote all pairs (a,b) till the end of the jth INC-MAC-AUX (update) query made by A, such that B during its simulation made a call to F_x with input a and $F_x(a)$ returned b. Ultimately, while simulating the final Verify query of A (or even during the INC-MAC-AUX queries' step 1, which is essentially a verify query), we show that for some a and b determined by A, such that a has not been a query to $F_x(\cdot)$, the verify query returns 1 (and forall i $I_i \neq \perp$) iff $F_x(a)$ equals b. This claim proves the theorem.

We will follow notation from definition 2.3 for all the queries of adversary A.

We say that a node u was *assigned* a value by the algorithm PAT in the ith update query if this node is in the update path of the ith query. Clearly, the root is assigned a value in each update query, and a leaf is assigned a value in an update query only if it was the leaf being updated. For each node v, let $last(v,j)$ be the largest $i \leq j$ such that v was assigned a value in the ith query. If it was never assigned a value in an update query ($\leq j$) then $last(v,j) = 0$. Let $latest(v,j)$ be the V value assigned to v by the algorithm INC-MAC-AUX (of PAT) in query $last(v,j)$; if v is a leaf node then $latest(v,j)$ is just the data assigned to that leaf in the $last(v,j)$ query. The initial MAC-AUX query will be considered as the 0th query. Fact 1(a) below follows from the optimized initialization of the authentication tree.

Fact 1: (a) For all u, $latest(u,0) = 0$.
(b) For $k \geq 1$, for all u if $last(u,k) = k$, then $latest(u,k) = k * n - \text{level}(u)$, where level($u$) is the distance of u from the root.
(c) For all u, j and k, if $last(u,j) \leq k$, then $latest(u,j) = latest(u,k)$.

Fact 2: For all j, $0 \leq j \leq q$, if $\sigma'_{j-1} = \sigma_{j-1}$, then $V(v) = latest(v,j-1)$.

Claim 3: for any non-leaf nodes u,v, $u \neq v$, for all t,t',
$latest(u,t) \neq latest(v,t')$, or
$latest(u,t) = latest(v,t') = 0$.

Proof: Suppose both values are non-zero. Let $last(u,t) = k \geq 0$, and $last(v,t') = k' \geq 0$. Then by Fact 1(b), $latest(u,t) = k * n - \text{level}(u)$, and $latest(v,t') = k' * n - \text{level}(v)$. It follows that if these two values are same then $\text{level}(u) = \text{level}(v)$, and $k = k'$. But that is impossible, as in the kth update query only one node at each level gets a new V value $\square$

In the algorithm INC-MAX-AUX, each non-root node in the update path and the sibling path is first verified (except for the leaf of the update path – see Step 1) before its mac value is updated. Call these collection of nodes in the jth $(1 \leq j \leq q)$ update query as S_j. We stress that that the node being updated is not in S_j. We will also call the final Verify query of adversary A, the $(q+1)$th query. We define S_{q+1} to be the collection of nodes in the path from the leaf being verified to the root.

Unless otherwise mentioned, whenever we refer to $V(u)$ for some node, we will mean the V value supplied for node u by adversary in the jth query. The same will hold for data(u).

Claim 4: Either
(a) for every j, $1 \leq j \leq q+1$, for all nodes $u \in S_j$, $V(u)$ (or data(u)) $= latest(u, j-1)$, or
(b) there exists a j, $1 \leq j \leq q+1$, and a node $v \in S_j$ such that $(V(v)||V(\text{parent}(v))||\lambda)$ is not in List_{j-1} (i.e. is not equal to the first entry of any pair in List_{j-1}).

Proof: Suppose (b) does not hold. Then we prove (a) by induction on j.

Base case $(j = 1)$. By Fact 1(a), $latest(u, j-1) = 0$. Hence, the only entries in List_0 are ones corresponding to arguments $(0||0||0)$ and $(0||0||1)$. Thus, if (a) does not hold then (b) must hold.

Suppose that the induction hypothesis holds for $j-1$.

We do a nested induction on the nodes of the update path. So consider a node u in S_j, but restricted to the update path. Suppose its supplied $V(u)$ is indeed same as $latest(u, j-1)$. We will show that for v being either children of u (and v in S_j), $V(v)$ (or data(v)) supplied in the jth query is indeed $latest(v, j-1)$,

Let $last(u, j-1)$ be $k \leq j-1$. If $k = 0$, then both its children v have $V(v) = 0 = latest(v, j-1)$. Otherwise u is in S_k. Now, since $last(u, j-1) = k$, neither of u's children v have $last(v, j-1) > k$. In fact, one of them , say $v1$, has $last(v1, j-1) = k$, and the other, say $v2$, has $last(v2, j-1) < k$ (i.e. $v2$ is in the sibling path in the kth query). Moreover, $v2$ is also in S_k. Thus, by induction, V value supplied for $v2$ in the kth query is $latest(v2, k-1)$, which by Fact 1(c) is same as $latest(v2, j-1)$.

On the other hand, V value assigned to $v1$ in kth query is $latest(v1, j-1)$. Thus, for both v (i.e. $v1$ or $v2$), $(latest(v, j-1)||latest(u, j-1)||\lambda)$ was inserted in $List_k$. Moreover, by Claim 3, and Fact 1(b), these are the only values in List_{j-1}, with middle value $latest(u, j-1)$. Now, supppose for one of these v (i.e. $v1$ or $v2$), $V(v)$ is not the same as $latest(v, j-1)$. But, by the claim that (b) does not hold we have that $(V(v)||latest(u, j-1)||\lambda)$ is in List_{j-1}, which leads to a contradiction.

Thus, for v being either children of u, $V(v)$ supplied in the jth query is indeed $latest(v, j-1)$. That completes the nested induction step.

But since, $V(r)$ supplied in the jth query is indeed same as $latest(r, j-1)$ (as the τ values are not altered by the adversary), the induction step is proven. $\square$

We are now ready to complete the proof of Theorem 1.

We point out again that we use notation from Definition 2.3. Let **CleanUpdate** be the event $\forall i \in [1..q]\ : I_i \neq \perp$, and **Verfied** be the event $Verify_x(j, a, \sigma', \tau_q) = 1$.

Let u correspond to the leaf at index j as specified in the final Verify query of A. If the event **CleanUpdate** happens then $M_j^q = latest(u, q)$. This follows from the fact that either the jth leaf was never updated, in which case $M_j^q = M_j^0 = latest(u, 0) = latest(u, q)$, or it was updated at $last(u, q) = k$ to be block a_k, in which case $M_j^q = a_k = latest(u, k) = latest(u, q)$.

Since, data a to be verified at leaf node u corresponding to block number j is different from M_j^q, and hence is different from $latest(u, q)$ Claim 4(a) does not hold. Hence Claim 4(b) must hold. Let j be the query and v be the node in S_j, such that $(V(v)||V(\text{parent}(v))||\lambda)$ is not in List_{j-1}. Let $M' = (V(v)||V(\text{parent}(v))||\lambda)$, and $\tau' = C(v)$, where $C(v)$ is the value supplied by the adversary A in the jth query. Thus if **CleanUpdate** and **Verified** happen with probability p, then in the experiment of Definition 2.1, $F_x(M') = \tau'$ happens with probability at least p as well.

$\square$

5. Optimizations

Since the PAT scheme used a function $F : \{0,1\}^m \times \{0,1\}^{\max\{l,d\}+l+1} \rightarrow \{0,1\}^c$, and the security of PAT required this function to be a secure MAC. It is well know that if F is a secure pseudorandom function family, then it is also a secure MAC (as in definiiotn 2.1) [2]. The question then boils down to building en efficient PRF from $\max\{l, d\} + l + 1$ bits to l bits.

First note that, and PRF is susceptible to birthday attacks, and hence it is secure only upto at most $c/2$ queries. This implies, that l need only be $c/2$.

Moreover, since the update requires computing MAC on each node on the path from the leaf to the root, as well as the MAC on the sibling nodes, a single MAC can be computed and stored for both the siblings (since they both have the same parent). Thus, the storage requirement for the MAC values is $c/2$ bits per node. The nonces also require $c/2$ bits per node. Thus, the authentication tree in this optimized version has the same storage requirement for internal nodes as that in a Merkle tree.

Note that d can still be c bits, if we compute a different C value for each leaf.

Finally, note that we still have to describe an efficient PRF from $3c/2$ bits to c bits. The $3c/2$ bits in the argument come from $c/2$ bits each of the two children nonces, and the parent nonce. It is well known [2] that given a universal hash function family h (with key $k1$) from d bits to c bits, and a PRF family G (with key $k2$) from c bits to c bits, that $F_{k2}(h_{k1}(\cdot))$ constitutes a PRF family (with key $k1, k2$) from $d + c$ bits to c bits.

Let $k1$ be c bits. Consider the Galois field GF(2^c). Then, given, a and b (both c bits), then $a + b * k1$ is a universal hash function family, with key $k1$. This follows directly from the algebra of GF(2^c).

In general, if $h1$ is an XOR-universal hash function family with key $k1$, from c bits to c bits, then $a + h1_{k1}(b)$, is a universal hash function family from $2c$ bits to c bits.

Thus, the above construction of PRF from $2c$ bits to c bits can be seen as a tweakable PRF (cf. [12]). A similar construction with PRPs (called IAPM [11]) was shown to be a tweakable block cipher by [12].

6. Alternative Schemes

This scheme is similar to the XECB-MAC([7]) and PMAC schemes ([4]), except that it is incremental. As before, each internal node has a C value and a V value. The V value is only computed during updates or verification (i.e. it is secret), and is not stored as a part of the tree (only C values are stored or returned as part of σ). The MAC is computed using a pseudo-random permutation P (i.e. it is a MAC on same sized data as the size of the MAC value).

Each leaf has a unique address. A slightly different MAC function is employed at the leaf nodes. The algorithm picks (during initialization) a function h from a universal hash family. This function h is used to hash the data value at each leaf, along with its address to be the input of the PRP P^{-1}.

The plaintext data being authenticated at the leaves are the C values of the leafs. The V values at the leaf node u are computed as $P^{-1}(h(C(u)||address(u)))$.

The V value at an internal node is computed by an xor-sum of the V values of all its children. The C label of internal nodes is computed by encrypting the V value of that node with P.

We observe that the V value of any internal node u (including the root) is the XOR-sum of the V values of all the leaf nodes under this internal node u. By keeping the C labels of the internal node, we assure integrity of a leaf by just checking the path from the leaf to the root. The "address sensitivity" is built into the leaves using the function h.

This scheme however has the drawback of having a critical path of a decryption followed by encryption in a parallel implementation.

Acknowledgments

The authors would like to thank Hugo Krawczyk, Tal Rabin and Shai Halevi for helpful discussions and suggestions.

References

[1] M. Bellare, R. Guerin and P. Rogaway, XOR MACs: New methods for message authentication using finite pseudorandom functions, Advances in Cryptology - Crypto 95 Proceedings, Lecture Notes in Computer Science Vol. 963, D. Coppersmith ed, Springer-Verlag, 1995.

[2] M. Bellare, S. Goldwasser, "Lecture Notes on Cryptogrpahy", http://www-cse.ucsd.edu/users/mihir/papers/gb.html

[3] M. Bellare, O. Goldreich, S. Goldwasser, "Incremental Cryptography with Applications to Virus Protection", Proc. STOC 1995

[4] John Black and Phillip Rogaway, "A Block-Cipher Mode of Operation for Parallelizable Message Authentication", Advances in Cryptology - EUROCRYPT '02, Lecture Notes in Computer Science, Springer-Verlag, 2002

[5] M. Blum, W. Evans, P. Gemmell, S. Kannan, M. Naor, " Checking the Correctness of Memories", Algorithmica, Vol 12, pp 223-244, 1994.

[6] E. Buonanno, J. Katz, M. Yung, "Incremental Unforgeable Encryption", Proc. FSE 2001, LNCS 2355.

[7] V.D. Gligor, P.Donescu, "eXtended Electronic Code Book MAC", http://csrc.nist.gov/encryption/modes/proposedmodes

[8] M. Bellare, R. Canetti, and H. Krawczyk, " Keying hash functions for message authentication", Advances in Cryptology—Crypto '96, 1996.

[9] Shai Halevi, "An observation regarding Jutla's modes of operation", http://eprint.iacr.org/2001/015/

[10] J. Håstad, "Message Integrity of IAPM and IACBC", http://csrc.nist.gov/encryption/modes/proposedmodes/iapm/integrityproofs.pdf

[11] C. S. Jutla, " Encryption Modes with Almost Free Message Integrity", Eurocrypt 2001, LNCS 2045.

[12] Moses Liskov, Ronald L. Rivest, David Wagner: Tweakable Block Ciphers, CRYPTO 2002: 31-46

[13] M. Luby, "Pseudorandomness and Cryptographic Applications", Princeton Computer Science Notes, Princeton Univ. Press, 1996

[14] R. Merkle, "A certified digital signature", Crypto 89, LNCS 435, 1989.

Appendix (Earlier Version)

A1. Introduction

Merkle authentication trees [14] have found numerous applications in incremental authentication, hashing, and other cryptographic protocols. In its usual form, a large amount of data which may get updated in an incremental fashion, and which requires authentication, is hashed in a tree fashion (for instance, by a universal one-way hash function [5], or by an H-MAC [8]). If the tree has n leaves, then any update requires $O(\log n)$ hash function applications. Similar number of operations are required for verification. For both update and verification to have this efficiency, all the internal node values need to be stored.

Note that when a leaf node's value gets updated, the hash function applications are inherently sequential, and cannot even be pipelined. In other words, to compute the value of the nodes in the new tree, all values of the children of a node must already be in place.

In certain applications, these $O(\log n)$ hash function applications can become prohibitively expensive, in light of the fact that the computations cannot even be pipelined. One such application is the checking of correctness of memory [5]. In this application, a storage device (e.g. RAM) may be vulnerable, open to attack from an adversary who could read and modify the memory. However, a tamper proof processor may have small amount of internal memory, which could store the value of the root of the authentication tree generated from the whole unprotected memory. This root value serves as the authentication tag of the current state of the whole memory. The authentication tree may itself be stored unprotected. A fast cryptographic processor inside the tamper proof processor is required to generate the updated authentication tree without substantially deteriorating the performance of the overall system. A similar situation arises in network attached storage.

In this paper, we design a new authentication tree, which allows for all the cryptographic operations in the update and the verification process to be parallelized. Akin to the Merkle tree, an update requires recomputing the values of a path from the updated leaf to the root, and the values of the neighboring nodes, resulting in $O(\log n)$ cryptographic operations. However, in the new authentication tree all the operations can be performed independently, and hence in parallel. This new authentication tree is defined in the symmetric key setting. In other words, both the authenticating party and the verifying party must share the same secret key. Moreover, the cryptographic operations performed in the new authentication tree are block cipher invocations, i.e. we prove our security results in the pseudo-random permutation generator model (and not the H-MAC model [8]). We use the "whitening" with an XOR-universal (a weaker form of pairwise independence) sequence idea from the recent integrity-aware parallelizable mode IAPM [11].

In the new authentication tree scheme, the authenticating party and the verifying party share two secret keys, one for the block cipher, and one for generating the XOR-universal sequence. The

XOR-universal sequence is easily generated given the key and the index in the sequence (e.g. by a single multiplication in a Galois field). Thus, this scheme is not applicable in the public key signature setting, where the root value is signed with the private key corresponding to the public key. Also, unlike some definitions of authentication codes, our security goal is only to assure target collision resistance.

We now describe, rather tersely, the key ingredient of the new scheme. An update in the value of a leaf, requires the authenticating party to choose a new nonce per node in the path from this leaf to the root. This nonce just needs to be fresh, and hence can be chosen in a sequence. Next, the authentication tree is updated by encrypting the nonce value at each node (and the value chosen by the adversary for the leaves) but after it has been "whitened" with the XOR-universal value generated from the nonce of its parent (and the relative position of the child). The encrypted value and the nonce are stored at the node as an integrity check.

Note that only about $2 * \log n$ block cipher invocations are required for the update (in a binary tree). Moreover, all the encryptions can be done in parallel, as the nonces can be chosen in parallel. Verifying that a leaf node has the correct (authenticated) value requires checking all the integrity checks (i.e. encryption with whitening) on the path from this leaf to the root. Of course, the root value is assumed to be unaltered. The verification steps can also be done in parallel.

We prove, modeling the block cipher as a random permutation, that an adversary has probability $O((s+n*\log s)^2 * 2^{-m})$ of succeeding in forging a tree whose root value is same as that of the given authentication tree, where s is the number of nodes in the tree, n is the number of updates, and m is the size of the block cipher. We prove our result in the adaptive adversarial model, where the adversary can choose the update leaves and their values.

In section A3, we address the problem of incremental authenticated encryption. The leafs in such a setting are allowed to store encrypted data, which must be authenticated by the tree in an incremental fashion. The encryption of the data is done using IAPM ([11]), with the modification that that the last block (authentication tag block) is pre-whitened using a nonce of the parent node of the leaf. This optimization reduces the critical path from two encryptions to one, in a parallel implementation.

A2. Definitions

Definition (*Authentication Tree*):

An Authentication Tree T is a structure $T = (S, r, L, D, V, C)$, where S is a set of nodes, with $r \in S$ being the root. The leaves are the set of nodes $L \subset S$. Let I be the internal nodes, i.e. $I = S - L$. The bijective map $D : I \times \{0,1\} \rightarrow S$ is the daughter map, with 0 signifying the left daughter, and 1 signifying the right daughter. V and C are labels on S. There is no C-label on the root node r. The depth of a node is given by the function d, with the root being at depth $d(r) = 0$, and the children of a node at a depth one more than that of the node.

We also have the obvious parent function $U : S - \{r\} \rightarrow S$.

The V and C labels of the nodes can not take arbitrary values. To this end, we define a consistent authentication tree.

Definition (*Consistent Authentication Tree (CAT)*): A Consistent Authentication Tree is $\mathcal{C} = (T, F, G)$, where T is an authentication tree, F is a random permutation , and G is an XOR-universal function (i.e. each $G(i)$ is uniformly random and $G(i)$ xor $G(j)$ is also uniformly random for $i \neq j$). Moreover the CAT $\mathcal{C}$ also satisfies the following:

- $\forall s, t \in I$, $s \neq t$, $V(s) \neq V(t)$
- $\forall s \in I$, $\forall j \in \{0, 1\}$: $C(D(s,j)) = F(V(D(s,j)) \oplus G(V(s);j))$

Here $V(s);j$ means, j appended to $V(s)$ as a bit string. Thus, if G is a function from m bits to m bits, we assume that the labels V are $m-1$ bits each. □

Of course, the authentication trees get updated, possibly by an adversary supplying new leaf values. We will consider a sequence of such updates as follows, as a setup for a subsequent collision attack by the adversary.

Definition An *update sequence of CATs* is a sequence of CATs, with the same F, G and S (and also same r, L and D). The sequence of CATs has the authentication tree sequence $T^1, T^2, ...T^n$, where T^{i+1} is obtained from T^i by replacing the labels of one of the leaves, and the labels on the nodes on the path to the root r, and the resulting change in the C-labels of the neighboring path (see below for a precise definition). Superscripts on V and C will denote the V and C-labels in the corresponding tree. Let the leaf replaced in T^i to obtain T^{i+1} be $l(i)$. The following additional consistency rules are required: $V^1(r)$ is greater than all other V-labels in T^1. Let the depth of the leaf $l(i)$ be d. Then, for each node $s \in I$ of depth $d' < d$ on the path from the root to the node $l(i)$ (i.e. excluding $l(i)$): $V^{i+1}(s) = V^i(r) + d - d'$. The label $V^{i+1}(l(i))$ of the updated leaf can take any value. For all other nodes (which are not on this path), the V label remains the same.

The nodes on the path from $l(i)$ to the root, and the neighboring nodes will be called $\bar{S}^{i+1}$. Lets call these nodes the *updated nodes* of round $(i+1)$. To be more precise, first define, for $i > 0$, $\hat{S}^{i+1}$ (the *updated path* of round $(i+1)$) to be the closure of the set $\{U(l(i))\}$ under the parent relation U. In other words, if $s \in S$ is in $\hat{S}^{i+1}$, then so is $U(s)$. As a base case, $U(l(i))$ is in $\hat{S}^{i+1}$. Also, define $\hat{S}^1 = I$. Then for $i = 0$ to $n-1$:

$$\bar{S}^{i+1} = \{s \in S - \{r\} | \exists j \in \{0,1\}, \exists t \in \hat{S}^{i+1} : D(t,j) = s\}$$

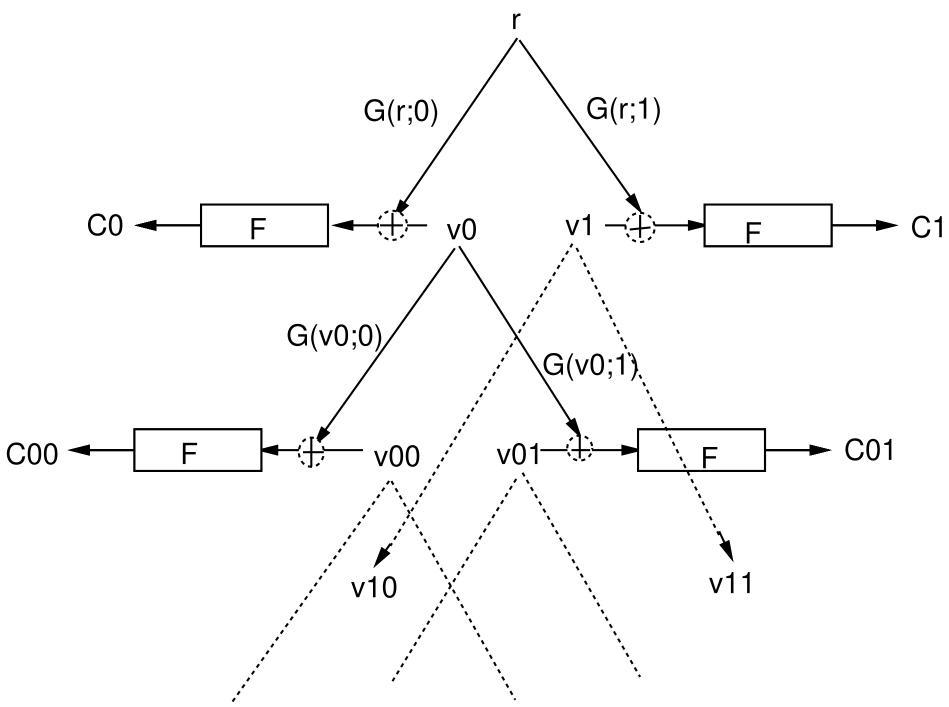

Figure 2: Two Levels of an Authentication Tree

Fact 1: Two nodes s and t (s maybe same as t) which are in $\hat{S}^i$ and $\hat{S}^j$ respectively, for $i \neq j$, will have different V-labels. In other words, $V^i(s) \neq V^j(t)$ if either $s \neq t$, or $s(=t) \in \hat{S}^i \cap \hat{S}^j$.

Definition A *Colliding tree* T' of an update sequence of CATs as above, is an authentication tree with the same S, r, L and D, and with the requirement that $V'(r) = V^n(r)$, i.e. the V-label of the root of the colliding(attack) tree is the same as the V-label of the root of the last tree in the update sequence, and there exists a leaf l such that $V'(l) \neq V^n(l)$, and all nodes s on the path from r to l (including r but excluding l) satisfy the consistency requirement:

$$C'(D(s,j)) = F(V'(D(s,j)) \oplus G(V'(s);j))$$

where j is such that $D(s,j)$ is the daughter of s on this path. Note that the colliding tree is not required to satisfy the conditions mentioned for a CAT (see the definition of CAT). In other words, the consistency requirement is only required to hold along the path from r to l.

Definition: The *success probability* of an adversary which chooses adaptively (with access to all trees till T^i) the update leaf $l(i)$ of each tree in the update sequence and the V-labels of these leaves, is given by the probability that it can generate a colliding tree as above.

A3. Incremental Encryption and Authentication

We now describe a further optimization when the data to be authenticated is also to be stored encrypted. In this optimization, the data at each leaf is encrypted using IAPM [11], except with a minor modification for generating the authentication tag block.

We first describe IAPM [11],[9],[10]. As in the authentication trees of previous sections, the encryption mode IAPM is built on two random functions: a random permutation F, and an independent XOR-universal function G. The idea that the same xor-universal function can be used in all messages was first pointed out in [9]. As before we will restrict our attention to XOR-universal functions implemented using multiplication in GF(2^m).

To encrypt the ith message P^i, with generic blocks named P^i_j, pre- and post- whiten the blocks using G, and encrypt using F, i.e. the ciphertext block C^i_j is given by

$$C^i_j = G(j;i) \oplus F(P^i_j \oplus G(j;i))$$

where pre-determined number of bits are assigned for i and j in the argument to G. Essentially, the argument to G will never be repeated, even across different messages. Here j ranges from 1 to the number of blocks in P^i (say L). An authentication tag is also generated, as the $L+1$th block of ciphertext as follows:

$$C^i_L = G(0;i) \oplus F(checksum \oplus G(L+1;i))$$

where *checksum* is the xor-sum of all the plaintext blocks in the ith plaintext message P^i.

A simple implementation of incremental authenticated encryption would just use IAPM as described for encryption of fixed size blocks at each leaf, and then authenticate the authentication tag C^i_L by making it the V-label of the leaf in the parallelizable authentication trees described in section A2. However, this leads to a critical path of two encryptions in the parallel implementation of the combined scheme. In the appendix we describe how to circumvent this problem by a simple optimization which also saves space, along with detailed proofs of the correctness of the optimized scheme.

We describe how the encryption scheme IAPM is incorporated in the new incremental encryption and authentication scheme.

There will be a fixed number of plaintext blocks assigned to each leaf in the authentication tree. The argument to G, which we have also called nonce previously, will be assured to be fresh. Essentially, when the data corresponding to a leaf is modified, it is encyrpted using a new nonce for the leaf (for whitening purposes). It is called VL in Figure 3. As in section A2, all the nodes on the path from this leaf to the root also get new nonces. To generate the last block (as in IAPM – the authentication tag block), the checksum of the data is first whitened with a value generated from the nonce of the parent of the current leaf (called V0* in fig 3), followed by block encryption, and then post-whitened with a value generated from the nonce of this leaf.

The nonce for the leaf, and this encrypted tag value is stored as V and C labels respectively. The encrypted data is stored as additional ciphertext labels $e[\cdot]$ for the leaf nodes.

As before, a colliding CAT C' has the same V label for the root as the last tree in the update sequence. However, now any one of the labels $e[i]$ for some leaf l maybe different from that of the same label in tree T^n.

On an update, the V label of the leaf is also chosen in a sequence now. Thus, $V^{i+1}(s) = V^i(r) + 1 + d - d'$, for each node $s \in S$.

The adversary is adaptive and has chosen plaintext power in the first phase in the sense that it chooses which leaf is to be updated in each round, and also provides the plaintext for all the blocks in that leaf.

Let's assume that each leaf has B blocks of data stored with it. The encrypted data will be stored in B nodes which will be children of the leaf nodes (the leaf node is now a misnomer). For a leaf l, these nodes will be referred to by $l[k]$ (for $k \in [0..B-1]$). Each such node will have a label e.

Then, the plaintext $p^{i+1}[k]$ ($k \in [0..B-1]$) provided in round $i+1$, generates the labels $e(l(i)[k])$

as follows (recall leaf $l(i)$ is updated to generate the tree T^{i+1}):

$$e(l(i)[k])^{i+1} = F(p^{i+1}[k] \oplus G(V^{i+1}(l(i)); k)) \oplus G(V^{i+1}(l(i)); k)$$

Finally, the C label for any leaf l is given by:

$$C^{i+1}(l) = F(\text{checksum} \oplus G(V^{i+1}(U(l); j))) \oplus G(V^{i+1}(l); B)$$

where j is such l is the jth daughter of $U(l)$, and checksum is the xor-sum of all the B plaintext blocks at leaf l. This label will also be called the *authentication tag*. Note that, only leafs in the set of updated nodes may have its authentication tags modified. The string concatenation in the argument to G reserves $\lceil \log B \rceil$ number of bits for the second argument.

The S-sequence in figure 3 is essentially $G(VL; k)$, with k varying from 0 to $B-1$.

The nodes $l[k]$ will also be called *data blocks*.

Theorem 2: The success probability of any adversary in the above game is at most $\mathrm{O}(2 + (|S| * B + n * (2d+B))^2/2^m)$, where F and G are m-bit functions, B is the number of blocks stored at each leaf, $|S|$ is the number of nodes in the tree whose depth is d, and n is the number of updates.

Before we prove this theorem, we define a few events and some intermediate variables, and prove some lemmas.

We will use $E^i(l[k])$ for the random variable corresponding to $e(l[k])$ at node $l[k]$ in tree T^i (for $k \in [0..B-1]$). We will use $C^i(l)$ for the authentication tag label as before.

The M variables corresponding to these nodes are defined as $M^i(l[k]) = P^i(l[k]) \oplus G(V^i(l); k)$. We also introduce random variables N for

$$N^i(l[k]) = E^i(l[k]) \oplus G(V^i(l); k)$$

For internal nodes N will simply be just c. For the leaf nodes, $M^i(l) = checksum \oplus G(V^i(U(l); j)$, and $N^i(l) = C^i(l) \oplus G(V^i(l); B)$.

The proof is similar to that of Theorem 1, though more involved. In particular,event PD will be generalized now to include the pairwise distinctness of N and M variables of the data blocks.

The *updated data* $\hat{D}^{i+1}$ will be the data block nodes at leaf $l(i)$.

We will use C^i to refer to the C labels at all the internal nodes, the leaf nodes and the data block nodes. Similarly, c will be used to represent such a constant.

For any c and fixed value of G (say g), consider the event PD(c,g) (pairwise different):

Event PD(c,g):

$$\forall i,j \in [1..n] \forall s \in \bar{S}^i \cup \hat{D}^i, \forall t \in \bar{S}^j \cup \hat{D}^j, (i,s) \neq (j,t) : (M^i(s) \neq M^j(t)) \ \wedge \ (N^i(s) \neq N^j(t))$$

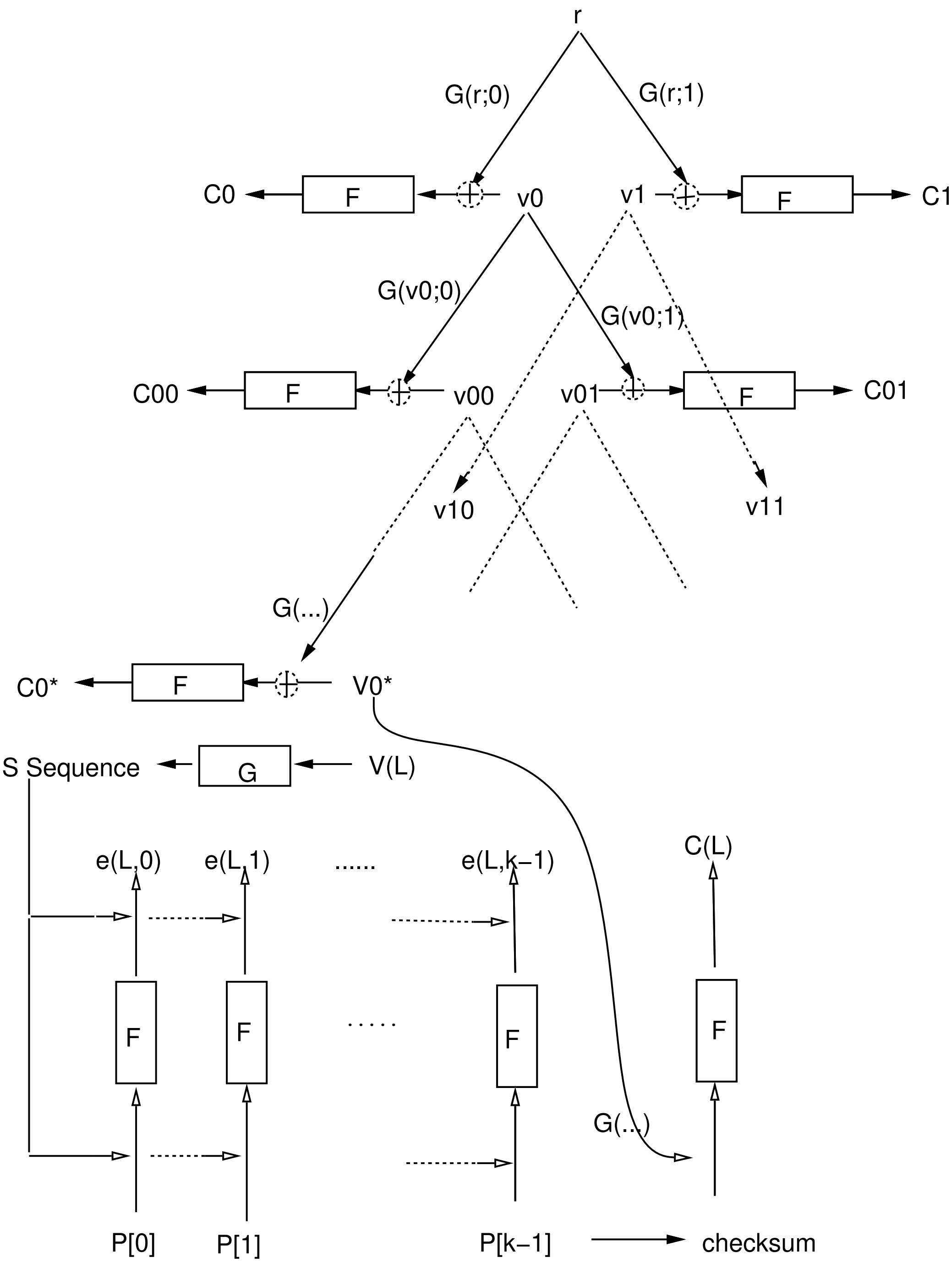

Figure 3: Incremental Encryption and Authentication

Thus, $\mathrm{PD}(c,G)$ is a random variable, and so is $\mathrm{PD}(C(F,G),G)$. We will denote the latter by just PD.

Lemma 5: for every constant c , and for any function g such that $\mathrm{PD}(c,g)$:

$$\mathrm{P}r[G=g|C(F,G)=c\wedge \mathrm{P}D(c,G)] = \frac{\mathrm{P}r[G=g]}{\mathrm{P}r[\mathrm{P}D(c,G)]}$$

Proof: Now

$$\Pr[G=g|C(F,G)=c \ \wedge \ \ \mathrm{PD}(c,G)] = \frac{\#F,G\ :\ G=g\wedge C(F,G)=c\wedge\ \mathrm{PD}(c,G)}{\#F,G\ :\ C(F,G)=c\wedge\ \mathrm{PD}(c,G)}$$

We now consider g such that $\mathrm{PD}(c,g)$ holds. Looking at the conditions in the numerator, since C is fixed to c, v is fixed (as the v variables are defined by c), and p is fixed, and fixing G to g fixes the M variables to a single value (with all Ms different as $PD(c,g)$ holds). Similarly, all the Ns are distinct. Since F is a permutation, there is exactly one F which satisfies the conditions in the numerator above. For the denominator, for each G such that $\mathrm{PD}(c,G)$ holds, there is exactly one F satisfying the condition in the denominator.

Thus,

$$\begin{aligned}
&\Pr[G=g|C=c\ \wedge\ \ \mathrm{PD}(c,G)]\\
&= \frac{1}{\#G:\ \mathrm{PD}(c,G)}\\
&= \frac{\Pr[G=g]}{\Pr[\mathrm{PD}(c,G)]}
\end{aligned}$$

□

W.l.o.g. assume that the path different in the colliding CAT T' from T^n is the leftmost path (while $V^n(r)=V'(r)$), and let the leftmost leaf be called z. We will also call the leftmost path the *attack path.*

Consider the following three events E1,E2 and E3.

Event E1: There exists a node $s\in S-\{r\}$ on the path from r to z (the leftmost leaf), such that for all $i\in[1..n]$, and all $t\in(\bar{S}^i-\{r\})\cup\hat{D}^i$, $M'(s)\neq M^i(t)$.

In other words, there is a node on the attack path, which has its M labels different from the M labels of every updated node in every round (including the M labels of updated data blocks).

Event E2: For some node s from z to root r: $N'(z)\neq F(M'(z))$

Event E3: there is an $x\in[0..B-1]$ such that

1. $\forall i \in [1..n] \forall t \in (\bar{S}^i - \{r\}) \cup \hat{D}^i: \ N'(z[x]) \neq N^i(t)$,
2. $\forall y, y \neq x, y \in [0..B-1]: \ N'(z[x]) \neq N'(z[y])$ and
3. $N'(z[x]) \neq N'(z)$

The condition E3 is similar to the condition required to prove the security of message integrity of IAPM.

Lemma 6:

$$\Pr[\neg(E1 \vee E2 \vee E3) | C = c \wedge PD(c,G)] \ \leq \ (2^d * (1+B) + n * (2*(d-1)+B)) * \frac{2^{-m}}{\Pr[PD(c,G)]}$$

Proof: With c fixed, the values v and p are also fixed. Similarly, the value c', v', and the attack path is fixed. Based on c', v', c and p, we show that one of the events E1, E2 or E3 is highly likely.

Let s be the node closest to the root (if any), such that $V^n(s) \neq V'(s)$. If such a node s exists, and s is not z then we show that $E1$ is highly likely. If such a node exists and it is z, then depending on whether $V'(z)$ has been used as a V somewhere else before either event $E2$ or $E1$ is highly likely. We give more details below in the proof. If even z has the same V label, then one of the data blocks is different, and event E3 happens with high probability.

(1) So, let $s \neq z$ be a node (if any) closest to the root such that $v^n(s) \neq v'(s)$. Let u be the parent of s. Since, $M'(s) = G(v'(u);0) \oplus v'(s)$, and $v'(u) = v^n(u)$, we have, $M'(s) = G(v^n(u);0) \oplus v'(s)$. Now, we calculate the probability that $M'(s) = M^i(t)$, the latter being $G(v^i(U(t));j) \oplus v^i(t)$, where j is such that $t = D(U(t),j)$, if t is an internal node; $G(v^i(l);z) \oplus P^i(l[z]$ if t is a data block with $t = l[z]$; and $G(v^i(U(t);j) \oplus$ checksum if t is a leaf.

Now,

$$\begin{aligned}
&\Pr[G(v^n(u);0) \oplus G(v^i(U(t));j) = v'(s) \oplus v^i(t) | C = c \ \wedge \ \mathrm{PD}(c,G)] \\
&= \ \Pr[G(v^n(u);0) \oplus G(v^i(U(t));j) = v'(s) \oplus v^i(t) | C = c \ \wedge \ \mathrm{PD}(c,G)] \\
&= \ 2^{-m} * \frac{1}{\Pr[PD(c,G)]}
\end{aligned}$$

The last equation follows by Lemma 5, unless $i = n$, $j = 0$, and $U(t) = u$ (this implies $t = s$), in which case the probability is zero if $v'(s)$ is different from $v^n(s)$, and one if the they are same. But the latter case cannot happen by the above assumption on s. The other cases follow similarly.

(2) Let z be the node closest to the root such that $v'(z) \neq v^n(z)$, and z is a leaf.

There are two cases. (a) Suppose $v'(z)$ was used as an index for whitening data block at some leaf at some time, say leaf l in tree T^i. Then, either for all data block indices k, $e^i(l[k]) = e'(z[k])$, or event E3 holds with high probability, with a block index x which violates the above. The latter is proved as in IAPM. In the former case, we have $checksum'(z) = checksum^i(l)$. Moreover, for the parent u of z, we have $v'(u) = v^n(u)$. And by design , $v^n(u) \neq v^i(U(l))$. Now, if $checksum^i(l) = checksum^n(z)$, then $M'(z) = M^n(z)$, and hence $F(M'(z)) = F(M^n(z)) = c^n(z) \oplus G(v^n(z); B)$ is unlikely to be $N'(z)$, as $N'(z) = c'(z) \oplus G(v'(z); B)$. In other words, event E2 happens with high probability.

If $checksum^i(l) \neq checksum^n(z)$, then $M'(z) \neq M^n(z)$, and it follows as in (1) that event $M'(z)$ is distinct from all other M with high probability, and hence event E1 happens with high probability.

(b) If $v'(z)$ was never used as an index for whitening data blocks, then again event E3 holds with high probability.

3) If there is no node s on the path from z to r such that $v'(s) \neq v^n(s)$, then, there must be a data block x such that $e'(z[x]) \neq e^n(z[x])$. In such a case, E3 holds with this x.

The lemma follows by noticing that each update causes at most $2(d-1) + B$ nodes to have their labels altered, where d is the depth of the leaf nodes. These $2(d-1)$ nodes include the nodes from the leaf to the root (excluding the root), and the neighboring path.

□

Lemma 7: For every constant c, with components of c restricted to S pairwise different,

$$\Pr[\neg\mathrm{PD}(c, G)] < 2 * (2^d * (1 + B) + n * (2 * (d-1) + B))^2 * 2^{-m}$$

Proof: Given a fixed c, the labels v and plaintexts p are also fixed, and hence the M labels are only a function of G. Thus the probability of any individual relation $M^i(s) = M^j(t)$ holding is 2^{-m}. This can be seen as follows. Let i', j' be such that $s = D(U(s), i')$ and $t = D(U(t), j')$, when s and t are in $\bar{S}$. Then, by definition V-labels of $U(s)$ and $U(t)$ are different (see Fact 1). When s or t is in $\bar{D}$, again an updated data block requires a new v.

As for N, the N labels are distinct for internal nodes by restriction on c. The N variables on data block nodes are distinct by argument similar to that of M values. The N labels on leaf nodes are distinct either because a different v is used, or if the same v is used because the c values are distinct by hypothesis.

The result follows by the union bound. □

We will denote c such that the components of c restricted to S are pairwise different by c *p.d.*

Lemma 8:

$$\Pr[\mathrm{PD}] > 1 - 2 * (2^d * (1 + B) + n * (2 * (d-1) + B))^2 * 2^{-m}$$

Proof:

$$
\begin{aligned}
& \Pr[PD] \\
= \;& \sum_c \Pr[PD \wedge C = c] \\
= \;& \sum_c \frac{\#F, G : C = c \wedge \mathrm{P}D(c, G)}{\#F * \#G} && (1) \\
= \;& \sum_c \frac{\Pr[PD(c, G)]}{\#F} && (2) \\
\geq \;& \min_{c\; p.d.}\{\Pr[PD(c, G)]\} && (3) \\
\geq \;& 1 - 2 * (2^d * (1 + B) + n * (2 * (d - 1) + B))^2 * 2^{-m} && (4)
\end{aligned}
$$

Equations (8) and (9) follow as in Lemma 5. Inequality (10) follows as the number of c (with components of c pairwise different) is at least as big as the number of F, as F is a permutation. Inequality (11) follows from lemma 7. □

Now we are ready to prove Theorem 2.

Proof (Theorem 2:)

We will partition the set of constants c into two sets S1 and S2. As in Lemma 6, the set of constants c which cause event E1 or E2 to happen with high probability will be in set S1. The constants c which cause E3 with high probability will be in set S2.

Then,

$$
\begin{aligned}
& \Pr[\neg(E2) \;\wedge\; \mathrm{PD}\,] = \sum_c \Pr[\neg(E2) \;\wedge C = c \;\wedge\; \mathrm{PD}\,] \\
= \;& \sum_{c\; p.d.} \Pr[\neg(E2) | C = c \wedge PD(c, G)] * \Pr[C = c \wedge PD(c, G)] \\
\leq \;& \sum_{c\; p.d,\; c \in S1} (\Pr[\neg(E2) | E1 \wedge C = c \wedge PD(c, G)] + \Pr[\neg(E1 \vee E2) | C = c \wedge PD(c, G)]) * \Pr[C = c] \\
+ \;& \sum_{c\; p.d,\; c \in S2} (\Pr[\neg(E2) | E3 \wedge C = c \wedge PD(c, G)] + \Pr[\neg(E3) | C = c \wedge PD(c, G)]) * \Pr[C = c] \\
\leq \;& \sum_{c\; p.d.} ((2^d * (1 + B) + n * (2 * (d - 1) + B)) * 2^{-m} + 2^{-m+1}) * \frac{\Pr[C = c]}{\Pr[PD(c, G)]} && (5) \\
\leq \;& 2 * (1 + 2^d * (1 + B) + n * (2 * (d - 1) + B)) * 2^{-m} && (6)
\end{aligned}
$$

Inequality (13) follows by lemma 7.

Inequality (12) follows by Lemma 6, and the following reasoning.

If event E1 happens, let s be the node such that its M value is different from all other M values, i.e. $M'(s) \neq M^i(t)$, for all all i and t. Then F being a random permutation, $F(M'(s))$ can take any value not already assigned, independent of G and $C'(s)$ (recall that $C'(s)$ was a function of C, which in turn was determined by G and values already assigned by F). Thus probability , under E1, that $N'(s) = F(M'(s))$ is at most 1 in $2^m - 2^d * (1 + B) - n * (2 * (d - 1) + B)$.

Similarly, if event E3 happens, probability that $M'(z) = F^{-1}(N'(z))$ is 1 in $2^m - 2^d * (1 + B) - n * (2 * (d - 1) + B)$, where z is the leaf node, as $M'(x)$ is a random variable whose value can be chosen after all other $F^{-1}(N')$ values have been chosen, and independently of G. Since $M'(z)$ can be expressed in terms of a function of G, and all other M' at that leaf, the result follows.

$\square$