A Tree-based Model of Unicast Stream Authentication

Goce Jakimoski
Computer Science Department, 253 Love Bldg,
Florida State University, Tallahassee, FL 32306-4530, USA

Yvo Desmedt
Department of Computer Science, University College London
Gower Street, London WC1E 6BT, United Kingdom

Abstract

When proving the security of a message authentication scheme, the messages are considered to be atomic objects. Straightforward application of such schemes to some information resources may introduce security flaws. Gennaro and Rohatgi (Crypto '97) identified the streams of data as an important class of information resources that can not be considered to be message-like, and they proposed a solution to the problem of stream signing when the stream is not known in advance.

The disadvantage of digital signing streams of data is that it is not efficient when non-repudiation is not important, as in the case of point-to-point communications. We present several schemes and also a family of schemes for stream authentication in a unicast setting. Since many authentication schemes have been broken, we will prove our solutions.

Keywords: stream authentication, MACs, stream signatures, unforgeability

1 Introduction

Streams of data, such as audio or video transmission, data feeds, etc., are now quite common on the internet. A stream is a bit sequence of finite, but a priori unknown length that a sender sends to one or more recipients [15, 4]. The difference between a stream and a message is that the stream must be processed as it is received. Streams occur naturally when the buffer/memory is shorter than the length of the message, or when on-line processing (e.g., by a human receiver) is required. Therefore, streams are usually divided into chunks that can be easily manipulated.

Gennaro and Rohatgi [15] were the first to propose a signing scheme for streams of data. Their scheme is based on a chain of one-time signatures [8, 23, 24] and provides both authenticity and non-repudiation. Several multicast stream authentication schemes based on similar concepts were proposed subsequently [32, 10, 1, 31, 4, 27, 25].

In many applications on the internet, non-repudiation is not necessary. A typical example is peer-to-peer stream communication. Moreover, besides applications like audio or video, interactive communications can also be described as a stream generated by two parties. E-mail conversations, chat, etc. are such examples. While the problem of multicast stream authentication setting has been extensively studied in the past years, the problem of peer-to-peer (or unicast) stream authentication (without non-repudiation) has not been studied formally. Clearly, one can use a multicast stream authentication scheme in a unicast setting. However, this approach is not very efficient since multicast schemes require frequent change of keys (i.e., one key is used to authenticate a relatively small group of packets).

Our contributions

In this paper, we adapt the notions of existential forgery, unforgeability and exact security [17, 2] to the case of peer-to-peer stream authentication.

We present several peer-to-peer stream authentication schemes. To prove the security of these schemes we introduce the concepts of stream authentication tree and stream authentication forest. We prove the security of a family of peer-to-peer stream authentication schemes based on authentication forest.

2 Unforgeable Stream Authentication

In a stream signature scheme, the data sequence is divided into smaller pieces we refer to as chunks, and an authenticator is assigned to each chunk. The decision whether a particular chunk is accepted as authentic is made based on the portion of the stream up to that chunk and the authenticator associated with that chunk. Thus, the delay between the moment when the chunk is received and the moment when it is accepted is reduced allowing for the receiver to consume the incoming bits almost as they arrive.

We must be able to distinguish between the end of the stream and the state that data is continuing to arrive. This strategy allows the receiver to know whether the adversary has shortened the message by deleting the last chunks. Note that this security property is not achieved by Gennaro-Rohatgi [15].

A stream signature scheme consists of a key generation algorithm G, a signing algorithm S and a verifying algorithm V. The key generation algorithm G outputs a pair of keys (s, p). The signing algorithm S takes as input a key s and a stream $\mathbf{M} = (M_1, \dots, M_l)$. Based on the key s and $(M_1, \dots, M_i)$ it computes a μ_i. We call the l-tuple $\boldsymbol{\mu} = (\mu_1, \dots, \mu_l)$ the *stream signature* of $\mathbf{M}$. $\mu_i (i \leq l)$ is called a *temporary signature*.

On input a key p, $(M_1, \dots, M_i)$, and $(\mu_1, \dots, \mu_i)$, algorithm V outputs i type/authenticity pairs $((b_1, v_1), \dots, (b_i, v_i))$, where the *type bits* $b_j \in \{0, 1\}$ are used to indicate whether the corresponding chunk is the last chunk in the stream or not, and the *authenticity bits* $v_j \in \{0, 1\}$ indicate whether the chunk is valid (accepted) or not. We say that the string $M_1 || \dots || M_i$ is *accepted as partial stream* if the stream $M_1 || \dots || M_{i-1}$ is accepted as a partial stream, the type bit b_i is zero and the authenticity bit v_i is one (authentic). The string $M_1 || \dots || M_i$ is *accepted as complete stream* if $M_1 || \dots || M_{i-1}$ is accepted as a partial stream (or i equals one), the type bit b_i is one and the authenticity bit v_i is one. If the stream $M_1 || \dots || M_i$ is not accepted, then it is *rejected*. It is obvious that if the stream $M_1 || \dots || M_i$ is rejected or accepted as complete, then the streams $M_1 || \dots || M_j (i < j \leq l)$ are rejected. A formal definition follows.

Definition 1 (Stream Signature Scheme) *A* stream signature scheme *is a triple* (G, S, V) *of probabilistic polynomial-time algorithms satisfying the following conditions:*

1. *The algorithm* G *(called the* key generator*) outputs a pair of bit strings* (s, p).

2. *On input a string* s *and* $(M_1, \dots, M_i)$*, where* $M_i \in \{0, 1\}^+$ *the* signing algorithm S *outputs* $\mu_i \in \{0, 1\}^+$. *If the complete stream is* $\mathbf{M} = (M_1, \dots, M_l)$ *we call* $\boldsymbol{\mu} = (\mu_1, \dots, \mu_l)$ *the* stream signature*, where* $l \in \{1, 2, 3, \dots\}$.

3. *On input a string p, $(M_1, \dots, M_i)$ and $(\mu_1, \dots, \mu_i)$ $(M_i, \mu_i \in \{0,1\}^+)$, the* verification algorithm V *outputs i* type/authenticity pairs (b_1, v_1), …, (b_i, v_i), $b_j, v_j \in \{0,1\}$. *If there is a $k \in \{1, 2, \dots, i\}$ such that $b_k = 1$ or $v_k = 0$, then the authenticity bits $v_j (k < j \leq l)$ are zero, where l is the number of chunks M_i in the complete stream $\mathbf{M} = (M_1, \dots, M_l)$.*

4. *For every pair (s,p) in the range of G, and for every $\mathbf{M} = (M_1, \dots, M_l), M_i \in \{0,1\}^+, l \in \{1, 2, 3, \dots\}$, algorithms S and V satisfy*

$$\forall i : 1 \leq i < l \ \ \Pr[V(p, \mathrm{Pref}_i(\mathbf{M}), \mathrm{Pref}_i(S(s, \mathbf{M}))) = (\underbrace{(0,1), (0,1), \dots, (0,1), (0,1)}_{i-\text{pairs}})] = 1$$

$$\Pr[V(p, \mathbf{M}, S(s, \mathbf{M})) = (\underbrace{(0,1), (0,1), \dots, (0,1), (1,1)}_{l-\text{pairs}})] = 1$$

where $\mathrm{Pref}_i((x_1, x_2, \dots, x_l)) = (x_1, x_2, \dots, x_i)$. *The probability is taken over the internal coin tosses of S and V.*

The previous definition says nothing about the security of the scheme. In order to prove that a particular stream signature scheme is secure, we need first to define what it means for a stream signature scheme to be secure. We will follow the existential forgery and exact security approaches [2, 17] and we will consider the *private key (stream authentication)* case only. The security of a public key stream signature scheme can be defined in a similar manner. The only difference is that the adversary in a public key scheme has access to the verification key p.

An adversary for a stream authentication scheme is a probabilistic algorithm E with oracle access to a signing and a verifying algorithm for a random but secret key pair (s, p). The adversary can request a stream signature of an arbitrary stream $\mathbf{M}$ by writing $\mathbf{M} = (M_1, \dots, M_l)$ on a special query tape. The signing oracle computes the signature $\boldsymbol{\mu}$ and returns it to E. E can also ask the verifier whether $\boldsymbol{\mu}$ is a valid signature for $\mathbf{M}$ by writing $(\mathbf{M}, \boldsymbol{\mu})$ on a special query tape. The verifying oracle returns an l-tuple $((b_1, v_1), \dots, (b_l, v_l))$. We will assume that there is some limit L on the size of the streams that can be submitted to the signing and verification oracles.

The goal of the adversary is to trick the receiver into accepting some string $M_1 || \dots || M_k$ as a partial or complete stream when either the string was never signed before or it was signed before as a different type (e.g, it was signed as a complete stream, but now it is accepted as partial). In the first case, the adversary can change the contents of the stream. In the second case, by changing the type, the adversary can either cut the stream or trick the receiver into believing that the stream is longer than its actual size. We keep track of the strings that are already signed using a *set of queries* $Q_E^{S(s,\cdot)}$. For each signing query $(M_1, M_2, \dots, M_l)$, we add the stream/type pairs $(M_1, 0)$, …, $(M_1 || \dots || M_{l-1}, 0)$, $(M_1 || \dots || M_l, 1)$ to the set of queries $Q_E^{S(s,\cdot)}$. The adversary E is successful if it makes a verify query $((M_1, \dots, M_l), (\mu_1, \dots, \mu_l))$ such that $M_1 || M_2 || \dots || M_k$ is accepted (as partial or complete) for some $k \in \{1, 2, \dots, l\}$, and $(M_1 || \dots || M_k, b_k)$ not in the set of queries. In other words, the output of the verifier is $((0,1), (0,1), \dots (b_k, 1), (b_{k+1}, v_{k+1}), \dots, (b_l, v_l))$, but $(M_1 || \dots || M_k, b_k) \notin Q_E^{S(s,\cdot)}$. The pair $((M_1, \dots, M_l), (\mu_1, \dots, \mu_l))$ is called a *forgery* and the k-tuple $((M_1, \mu_1), \dots, (M_k, \mu_k))$ is called a *partial forgery*[1].

The attack on the stream authentication scheme is (q_s, q_v)-attack if the adversary makes no more than q_s signing queries and no more than q_v verify queries. If, in addition, E runs for no more than t steps, then the (q_s, q_v)-attack is a (t, q_s, q_v)-attack. We say that the adversary $[q_s, q_v, \epsilon]$-breaks the scheme if the attack is a (q_s, q_v)-attack and it is successful with at least ϵ probability.

[1]The term partial forgery is somewhat misleading since we use it even when the stream $M_1 || \dots || M_k$ is complete.

The adversary $[t, q_s, q_v, \epsilon]$-breaks the scheme if the attack is (t, q_s, q_v)-attack and the probability of success is at least ϵ. The scheme is $[t, q_s, q_v, \epsilon]$-unforgeable if there is no adversary that can $[t, q_s, q_v, \epsilon]$-break it. The formal definition is given below.

Definition 2 (Unforgeability) *The stream authentication scheme (G, S, V) is $[t, q_s, q_v, \epsilon]$- un-forgeable if for every probabilistic algorithm E that runs in at most t steps, and makes at most q_s queries to a signing oracle $S(s, \cdot)$ and at most q_v queries to a verification oracle $V(p, \cdot)$, it holds that*

$$\begin{aligned}\Pr[\exists i \in \{1, \ldots, l\} \text{ s.t.} \quad & V(p, (M_1, \ldots, M_l), (\mu_1, \ldots, \mu_l)) = ((0,1), \ldots, (b_i, 1), (b_{i+1}, v_{i+1}), \ldots, (b_l, v_l)) \\ \wedge \quad & (M_1 || \ldots || M_i, b_i) \notin Q_E^{S(s,\cdot)} \ ] \ < \ \epsilon\end{aligned}$$

where $b_j, v_j \in \{0, 1\}, 1 \leq j \leq l$, (s, p) is a key pair generated by G, $((M_1, \ldots, M_l), (\mu_1, \ldots, \mu_l))$ is E's output and $Q_E^{S(s,\cdot)}$ is the set of queries. The probability is taken over the coin tosses of G, S and V as well as over the coin tosses of E.

3 Some Practical Schemes

In this section we present three practical schemes.

3.1 SN-MAC

SN-MAC (sequence numbers and MACs) is a straightforward stream authentication scheme. There is a unique number N_s assigned to each stream (e.g., using counter). A sequence number i, which determines the position of the chunk within the stream, and a type bit b_i, which indicates whether the chunk is the last chunk of the stream or not, are assigned to each chunk of the stream. The chunks are then signed (authenticated) independently using some message authentication scheme (see Fig 1).

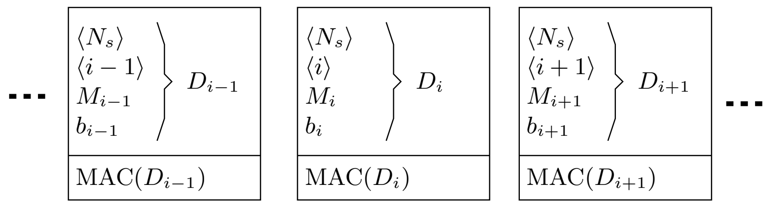


Figure 1: SN-MAC

Suppose that the data D_i in each packet i consists of three blocks $D_i = d_{i,1}||d_{i,2}||d_{i,3}$, and that a CBC MAC scheme is used for the computation of the authentication tags. In this case, the generation of the authentication tag $\text{MAC}(D_i)$ can be described using a labeled tree as in Fig 2. The blocks of D_i are assigned as labels to the external nodes (leaves) of the tree. A message label $\text{msg}[x]$ and a tag $\text{tag}[x]$ are assigned to each internal node x. The message label $\text{msg}[x]$ is a concatenation of the messages corresponding to the children of x, and it "keeps track" about what part of the stream is processed to get the tag $\text{tag}[x]$. The tag $\text{tag}[x]$ is either a function of the message label $\text{msg}[x]$ (e.g.,$\text{tag}[1], \text{tag}[2]$ and $\text{tag}[4]$) or a function of the tags of the children of x (e.g., $\text{tag}[3]$ and $\text{tag}[5]$). Note that only $\text{tag}[5]$ will be used for the verification of the authenticity of D_i. The rest of the tags are some intermediate results.

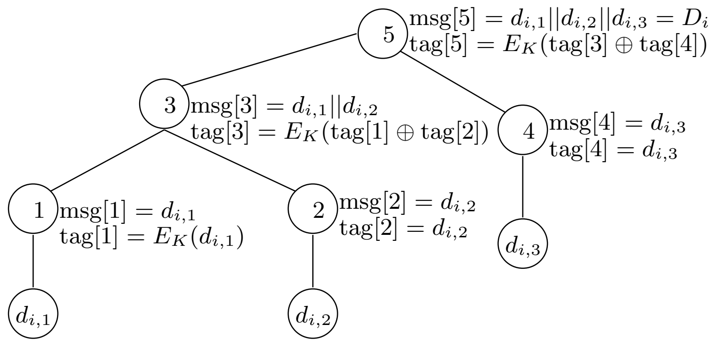


Figure 2: A tree-based description of the computation of MAC(D_i) in SN-MAC

3.2 ReMAC

In a ReMAC (recompute the MAC) scheme, the temporary signature associated with some chunk M_i is computed by signing the partial stream $M_1|| \dots ||M_i$, not just the chunk M_i (see Fig 3).

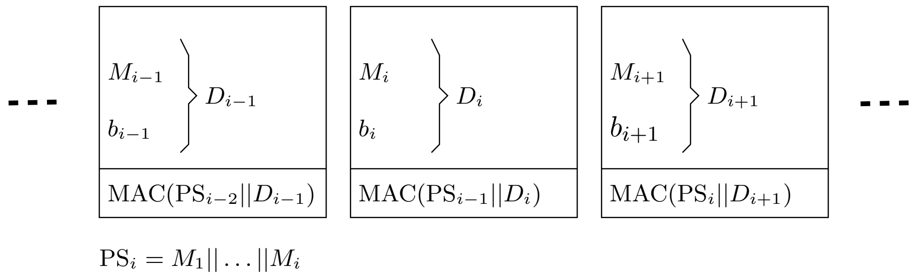


Figure 3: ReMAC

The tree-based description of ReMAC is given in Fig 4.

3.3 MACC

MACC (MAC chaining) scheme is depicted in Fig 5. An authentication tag is computed for each chunk M_i of the stream M and then the chunk M_i is chained to the previous chunk by computing MAC of the concatenation of the tags. It is assumed that σ_0 is an empty string.

A tree-based description of the tag generation in MACC is given in Fig 6.

4 A family of schemes

In this section, we generalize the design principles of the schemes introduced in Section 3 by introducing the notions of stream authentication tree and stream authentication forest. This generalized approach is motivated by the following arguments:

- We can avoid the problem of providing a separate security proof for each scheme presented in Section 3 by proving the security in the general case. Furthermore, we can select other practical schemes from the general class discussed here for our specific application. The security of these schemes trivially follows from the security of the general class.

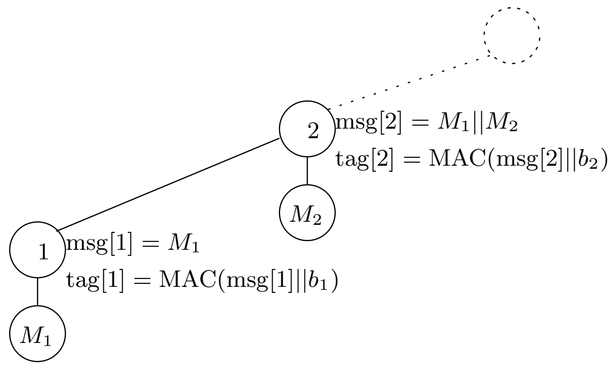
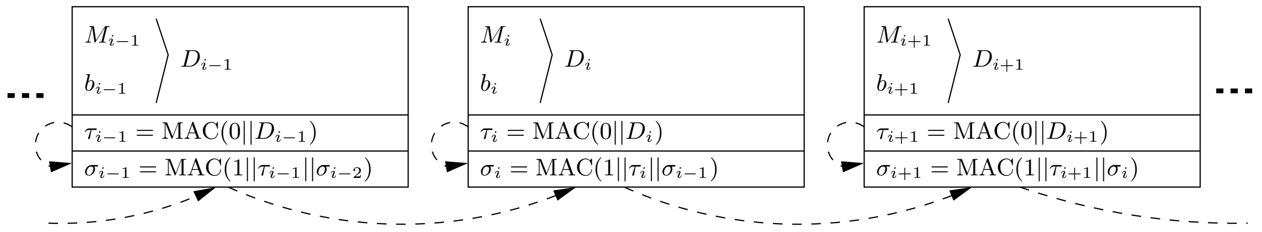


Figure 4: A tree-based description of the computation of authentication tags in ReMAC



M_{i-1}
b_{i-1}
D_{i-1}
$\tau_{i-1} = \mathrm{MAC}(0||D_{i-1})$
$\sigma_{i-1} = \mathrm{MAC}(1||\tau_{i-1}||\sigma_{i-2})$
M_i
b_i
D_i
$\tau_i = \mathrm{MAC}(0||D_i)$
$\sigma_i = \mathrm{MAC}(1||\tau_i||\sigma_{i-1})$
M_{i+1}
b_{i+1}
D_{i+1}
$\tau_{i+1} = \mathrm{MAC}(0||D_{i+1})$
$\sigma_{i+1} = \mathrm{MAC}(1||\tau_{i+1}||\sigma_i)$

Figure 5: MACC

- There are other applications besides the typical stream applications like audio or video, where a sequence of chunks (or messages) needs to be authenticated (e.g., exchange of related messages). We can use the stream authentication schemes as another level of abstraction that can be used to design and prove security of more complex constructions using reduction techniques instead of other methods, like formal methods for security protocol verification [9, 29].

- Message authentication is just a special case of stream authentication where the stream consists of only one chunk. There are number of MAC schemes (e.g., [3, 6, 7, 19]) that can be described using the notion of stream authentication tree introduced in this paper.

- The MAC schemes can be used to build secure stream authentication schemes. Therefore the analysis of the stream authentication schemes can help us better understand the desirable properties of MAC schemes. For example, the analysis in Section 5.2 shows that it is important if a MAC schemes is forgeable on collision or not. Whether the ReMAC scheme can be efficiently implemented or not depends on the underlying MAC scheme, etc.

An example of a stream authentication tree is shown in Fig 7. The pieces of the stream are assigned as message labels to the external nodes (leaves) of the tree. A message label $m(x)$, a tag $\tau(x)$, a tag type and an algorithm label A_x are assigned to each internal node x. We now explain this in more details.

The message $m(x)$ is a concatenation of the messages corresponding to the children of x starting from the first (left) child and ending with the last child. So, $m(x)$ is the concatenation of the pieces of the stream assigned to the leaves of the subtree rooted at x when traversed in postorder (postfix).

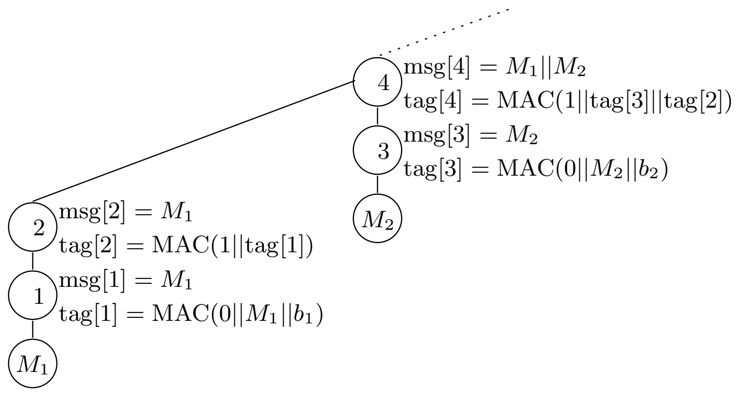


Figure 6: A tree-based description of the computation of authentication tags in MACC

We say that the external node (leaf) x in a tree T is a *leftmost leaf* iff every ancestor of x (including x itself) is the first (left) node among its siblings. The *leftmost path* is the path between the root and the leftmost leaf. The internal nodes in the leftmost path have a special role. Namely, the messages associated with some of these nodes are the partial (or complete) streams to be authenticated, and the corresponding tags are final results of some temporary signature computation.

We split the tag types based on two properties, being:

Whether the tag type is final tag or not. A *final tag* is defined as one that can be used to verify the authenticity of a (partial) stream. Otherwise it is called *intermediate*.

Whether the tag is a tag-of-tags, or a tag-of-chunks. We allow two ways to compute the tag $\tau(x)$ associated with an internal node x, either

- as a function of the message corresponding to x or,
- as a function of the tags of the children of x.

So, in total we can distinguish between 4 tag types, or internal nodes in the tree. We can now categorize the nodes of the tree as following:

Definition 3 Message nodes *are the external nodes of the tree. A piece of the stream is assigned as a message label to each message node and the concatenation of the message labels of the external nodes when visited in postorder gives the stream.*

Internal nodes *can be categorized as:*

Degenerated *Such a node is intermediate and is a tag-of-chunks.*

TPS (tag-of-partial-stream) *Such a node is a final node and is a tag-of-chunks.*

ITT (intermediate-tag-of-tags) *Such a node is intermediate and is a tag-of-tags.*

FTT (final-tag-of-tags) *Such a node is final tag and is a tag-of-tags.*

Definition 4 (Stream Authentication Tree) Stream authentication trees *are ordered trees. A message label is assigned to each external node (leaf) of the tree. A 4-tuple* $(m(x), \tau(x), t(x), A_x)$ *is assigned to each internal node* x*, where the string* $m(x)$ *is a message label, the string* $\tau(x)$ *is a*

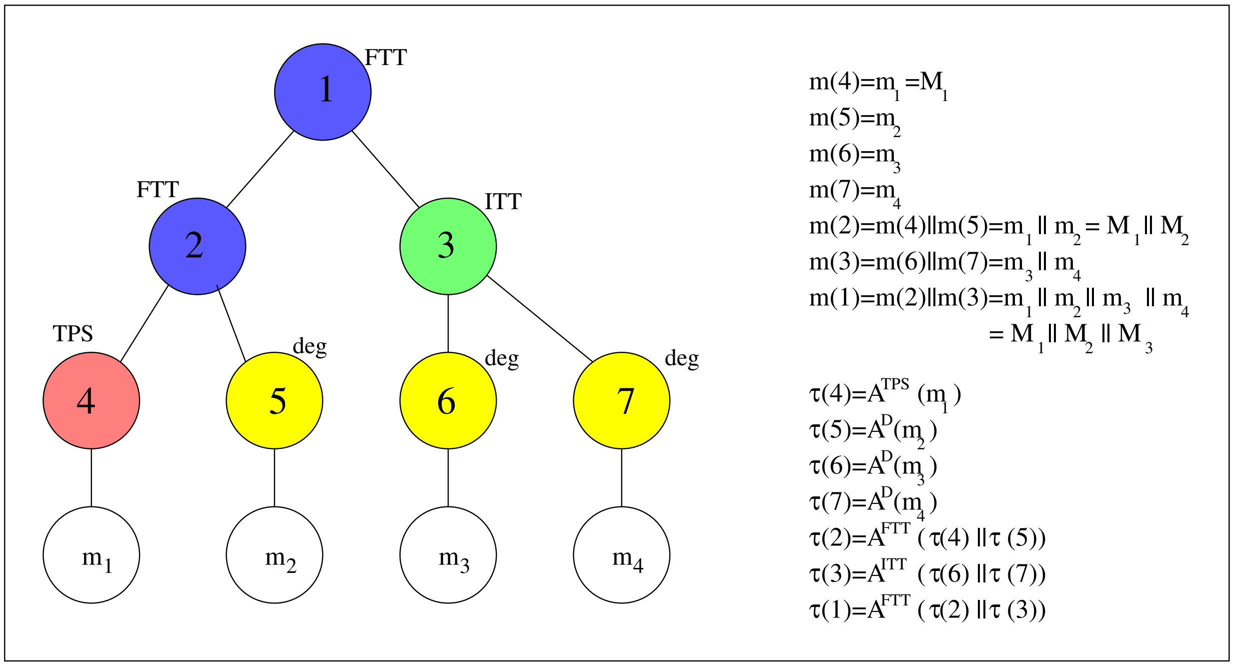


Figure 7: An example of a 4C stream authentication tree

tag, $t(x) \in \{\text{degenerated}, \text{TPS}, \text{ITT}, \text{FTT}\}$ *is a tag type, as defined in Definition 3, and* A_x *is an algorithm label that specifies the tag computation algorithm used at the node* x*. Let* $v_1, v_2, \ldots, v_k$ *be the children of an internal node* x*. The following properties must hold for the message and tag labels:*

$$m(x) = m(v_1)||m(v_2)||\ldots||m(v_k)$$

$$\tau(x) = \begin{cases} A_x(m(x)) & \textit{if } t(x) \in \{\text{degenerated}, \text{TPS}\} \\ A_x(\tau(v_1)||\tau(v_2)||\ldots||\tau(v_k)) & \textit{if } t(x) \in \{\text{ITT}, \text{FTT}\} \end{cases}$$

Note that the input to the tag computation algorithm is the concatenation of tags $\tau(v_1)||\ldots||\tau(v_k)$ instead of a tuple $(\tau(v_1), \tau(v_2), \ldots, \tau(v_k))$. Such definition is more convenient for the analysis presented below since all tag computation algorithms have exactly one input. We don't think that our choice is too restrictive since the individual tags can be extracted from the concatenated string (e.g., using known or fixed tag length).

Lemma 1 *The internal nodes have the following properties:*

1. *If* x *is a parent of an external node (leaf), then the tag type* $t(x)$ *cannot be* ITT *or* FTT*.*
2. *If* x *is not in the leftmost path, then the tag type* $t(x)$ *cannot be* TPS *or* FTT*.*

Proof. The first property follows straightforward from the definition since ITT and FTT are both tag-of-tags. The second follows from the fact that a partial stream must have the chunk M_1 in it. ■

In a stream authentication scheme based on the notion of a stream authentication tree, the signing algorithm first constructs a tree and assigns types and algorithms to the nodes of the tree, and then, it computes the authentication tags. A more formal description is given below. In most of the practical schemes, the structure of the authentication tree and the algorithms that will be used for the computation of the tags are known in advance, and they are not changed over time. Therefore, one can compute the authentication tags without having to construct a stream authentication tree.

We will refer to the labeled tree derived from a stream authentication tree by discarding the message, tag and algorithm labels of the internal nodes as the *structure* of the stream authentication tree. For each chunk M_i of the stream $(M_1, \dots, M_l)$, the signing algorithm of a *tree stream authentication scheme* first runs a (probabilistic) structure construction procedure. The structure construction procedure takes as input the structure constructed to compute the temporary signature for the partial stream $M_1||\dots||M_{i-1}$, the length of the chunk and information indicating whether the chunk is last or not, and outputs the "new" part of the structure that will be used to compute the temporary signature of the partial stream $M_1||\dots||M_i$. Clearly, the tag type of the new root must be either TPS or FTT, and the tag type of any other new node must be either degenerated or ITT. After the structure construction phase, the signing algorithm runs a (probabilistic) coloring procedure. The coloring procedure assigns algorithm labels to the new nodes. As mentioned before, the algorithm labels specify which algorithm will be used to compute the tag. The algorithm labels are selected from some set of algorithm labels (or colors) that is previously agreed upon between the sender and the receiver. The labels in the set can refer to both algorithms that use key and algorithms that do not use a key. If an algorithm label refers to an algorithm that uses a key, then it must specify the actual key that is used. For example, $\mathrm{CBC}-\mathrm{MAC}_{K_0}$ is a possible algorithm label and it specifies that the tag will be computed using CBC-MAC when the key is K_0. After the structure construction and the coloring, the signing algorithm computes the message and tag labels of the new nodes. Obviously, the newly constructed tree should be such that the message label of its root is $M_1||\dots||M_i$. The temporary signature of the partial stream $M_1||\dots||M_i$ consists of the tags of the new nodes[2] and the randomness used by the structure construction procedure, the coloring procedure and the tag computation algorithms.

Given the stream and the temporary signatures, the verification algorithm reconstructs the tree. The message label of a TPS or FTT node is accepted as valid only if the computed tags equal the tags that were sent and the previous partial streams are also valid. The set of all strings is partitioned into a set of complete and a set of partial streams (e.g., using end-of-text character). If the message label belongs to some predefined set of complete streams, then it is accepted as a complete stream. Otherwise, it is accepted as a partial stream.

The concept of stream authentication tree can be extended to stream authentication forest.

Definition 5 (Stream Authentication Forest) *A* stream authentication forest *is an ordered forest of stream authentication trees.*

In a *forest scheme*, a unique number N_s is assigned to each stream that is submitted for signing. A prefix to the message label of the leftmost leaf of each tree is also added. The prefix is a concatenation of the binary representations of the stream number N_s and the position j of the tree within the forest (analogously to SN-MAC scheme). We will refer to the scheme used to construct the trees of the forest as an underlying tree scheme.

[2]In order to speed-up the stream authentication, one can compute the tags in a distributed manner where each node computes a tag and sends the result to some other node. Since we have defined the temporary signature to include all "new"tags (not just the final tag), a distributed implementation of a secure scheme will remain secure even if the adversary can eavesdrop the communication between the nodes. However, in practice, we can send only the final tags and the randomness needed by the verifier to recompute them.

5 Security Analysis

In this section, we examine the security properties of the tree and forest stream authentication schemes. We show that we need at least four colors (different algorithms) in order to achieve security regardless of the structure of the stream authentication trees.

5.1 A particular approach

In Section 4 we did not specify restrictions on A_x. To facilitate proving general results we will now consider a simpler case. We will restrict A_x such that $A_x = A^{t(x)}$. Since we have four tag types, we could consider that we color the algorithms A_x based on these tag types.

We are primarily interested in deriving algorithms for $A^{t(x)}$ such that we can prove the resulting scheme to be secure regardless of the structure construction procedure that was used in the scheme.

In order to provide an answer, we consider the class of 4C (four color) schemes. An example of a 4C stream authentication tree is given in Fig 7. 4C schemes color the different node types (degenerated, TPS, ITT and FTT) using different colors. We denote by A^D, A^{TPS}, A^{ITT} and A^{FTT} the tag computation algorithms used at the degenerated, TPS, ITT and FTT nodes respectively. We will show that 4C forest schemes have *forgery or collision property* (defined further on), that is, if there is an adversary that can construct a forgery for the 4C forest scheme, then there is an adversary that can forge a tag or find a collision for some of the tag computation procedures A^{D}, A^{TPS}, A^{ITT} or A^{FTT}.

As usual, we assume that all algorithms are known to the adversary. Only the secret keys used by the sender and the receiver are unknown. From the discussion above, it follows that given the stream, the tags and the randomness, the adversary can reconstruct the stream authentication tree except for the secret keys that are used by the tag computation algorithms. Hence, hereafter, we slightly abuse the security model of the previous section and assume that the verify queries and the responses to signing queries are *shadowed* stream authentication trees that are constructed as follows. Each algorithm label that refers to an algorithm that uses a key is replaced by a label that specifies only the family, not the key. For example, the label $\mathrm{CBC}-\mathrm{MAC}_{K_0}$ in the stream authentication tree will become $\mathrm{CBC}-\mathrm{MAC}$ in the shadowed tree. We say that the shadowed tree rooted at a node x is *valid* if $m(x)$ is accepted when the tree is submitted to the verifier. The valid shadowed tree T is a *partial forgery tree* if the partial (resp., complete) stream corresponding to the root of the tree was not signed as partial (resp., complete) stream during some previous signing query. Let's consider a valid shadowed tree T. There is some input associated with each tag in the tree. For example, if the tag is a tag of a degenerated node, then the corresponding input is the message label of the node. If the tag is a tag of an ITT node, then the corresponding input consists of the tags of the children of the node, etc. The set of all such input/tag pairs is called a *set of authentic pairs corresponding to* T. The union of all sets of authentic pairs corresponding to trees constructed during some previous signing queries is called a *pool of known authentic pairs.* If an authentic input/tag pair is not in the pool of known authentic pairs, then we say that it is a *forgery pair.* Two input/tag pairs *collide* if the tags are equal but the inputs or the algorithms used to compute the tags are different. Given a node x in some stream authentication forest, a *stream authentication subforest at* x is the forest consisting of all the trees preceding the tree containing x and the subtree rooted at x. Let $((M_1, \mu_1), \ldots, (M_i, \mu_i))$ be some partial forgery and let x be the node where $M_1|| \ldots ||M_i$ is accepted. The subforest at x is called a *partial forgery forest* with *final node* x. Now, we can formally express the forgery or collision property (see Appendix A for the proof).

Lemma 2 (Forgery or Collision Property) *Let P_f be the set of authentic pairs corresponding to some 4C partial forgery forest consisting of the trees $T_1, \ldots, T_k$. At least one of the following statements is true:*

1. *There is a tree $T_i \in \{T_1, \ldots, T_k\}$ such that the tag associated with its root is forged.*

2. *There is a pair in P_f that collides with some known authentic pair.*

It is obvious that if we select the tag computation procedures so that it is hard to forge tags or find collision pairs, then the 4C forest scheme will be secure regardless of how we construct the trees. One such selection is considered in the next section where the tags are computed using MAC schemes.

The coloring scheme presented above not only provides security but also uses minimal number of colors. In Appendix B, *we present an example of structure construction procedure such that any scheme that uses the procedure and only three colors for the intermediate nodes can be broken.*

5.2 Security based on unforgeable MACs

We will introduce some new property for MAC schemes (see Definition 6) that will allow to prove our results. Note that we demonstrate that certain MAC schemes satisfy this new property.

We are going to show that a secure (unforgeable) 4C forest scheme can be obtained if a secure MAC scheme is used for tag computation. In particular, the tag computation procedures that we are going to consider are implemented as:

$$\begin{aligned}
A^{\mathrm{D}}(m(x)) &= \mathrm{MAC}(00||m(x)) \\
A^{\mathrm{TPS}}(m(x)) &= \mathrm{MAC}(01||m(x)) \\
A^{\mathrm{ITT}}(\tau(v_1)||\ldots||\tau(v_k)) &= \mathrm{MAC}(10||\tau(v_1)||\ldots||\tau(v_k)) \\
A^{\mathrm{FTT}}(\tau(v_1)||\ldots||\tau(v_k)) &= \mathrm{MAC}(11||\tau(v_1)||\ldots||\tau(v_k))
\end{aligned}$$

Definition 6 *A MAC scheme is* $[t, q_s, q_v]$-forgeable on collision *if there is an algorithm A that takes as input an initial finite set P of known authentic message/authenticator pairs and two pairs from P that collide, and outputs a forgery for the MAC scheme. The algorithm A runs in at most t time and it makes at most q_s signing and at most q_v verifying queries. If the two message/authenticator pairs are the only two pairs from P that collide, then the scheme is* $[t, q_s, q_v]$-forgeable on first collision.

Lemma 2 and the forgeability on collision property imply the following theorem.

Theorem 1 *Let E be an adversary that $[t', q'_s, q'_v, \epsilon]$-breaks a 4C forest scheme that uses a $[t'', q''_s, q''_v]$-forgeable on (first) collision MAC scheme for tag computation, and let $q = L(q'_s + q''_v + 1) + q''_s + q''_v$, where L is the maximum number of internal nodes that can appear in a forest. Then, there is an adversary that $[t' + t'' + cq, q, \epsilon]$-breaks (i.e., outputs a forgery) the underlying MAC scheme, where $c > 0$ is a small implementation dependent constant.*

The proof is given in Appendix C.

The following proposition gives an answer to the question whether secure MAC schemes that are forgeable on first collision exist.

Proposition 1 *PMAC and OMAC are $[c(|P| + l_M), 1, 0]$-forgeable on first collision, where l_M is the maximum message length, $|P|$ is the size of the set of known authentic pairs and $c > 0$ is a small implementation dependent constant.*

The description of PMAC and OMAC can be found in [7] and [19] respectively, and a sketch of the proof is given in Appendix E.

5.3 Security based on PRFs

Our result can be generalized for any MAC scheme if we assume that the function family defined by the signing algorithm of the MAC scheme is pseudorandom. From the forgery or collision property, it follows that forging a signature in a 4C forest scheme implies constructing a forgery or collision for the underlying MAC scheme. Therefore, if one can forge signatures in a 4C forest scheme with significant probability, then one can distinguish the function family defined by the signing algorithm of the MAC scheme from a random function family as explained below in more details.

In our analysis, we assume that there is some limit l_M on the length of the message that can be submitted for signing to the underlying scheme. Furthermore, the tags computed by the scheme are of fixed length l_T. The randomness part of the tag has length l_R (l_R is zero if the scheme is not probabilistic) and the rest of the tag is $n_T = l_T - l_R$ bits long. Let Σ_{l_M} be the set of all non-empty strings over the alphabet $\{0,1\}$ whose length is at most l_M, let $\mathcal{R}$ be the family of all functions from the set $\{0,1\}^{l_R} \times \Sigma_{l_M}$ (Σ_{l_M} in the non-probabilistic case) to the set $\{0,1\}^{n_T}$, and let $\mathcal{F}$ be the family of functions defined by the signing algorithm of the MAC scheme. A statistical test is a Turing machine A with access to an oracle $\mathcal{O}$. The oracle is selected to be a random function from $\mathcal{F}$ or a random function from $\mathcal{R}$ according to a random bit b. The algorithm A outputs 0 or 1. The advantage of distinguishing the finite family $\mathcal{F}$ from the finite family $\mathcal{R}$ is defined as

$$\mathrm{Adv}_A(\mathcal{F}, \mathcal{R}) = \frac{1}{2}(E[A^F] - E[A^R])$$

where $E[A^F]$ (resp., $E[A^R]$) is the probability that A will output 1 when the oracles are selected to be random functions from $\mathcal{F}$ (resp., $\mathcal{R}$). We say that A $[t, q, \epsilon]$-breaks $\mathcal{F}$ if it runs in at most t time, it makes no more than q queries, and it ϵ-distinguishes $\mathcal{F}$ from $\mathcal{R}$; that is $\mathrm{Adv}_A(\mathcal{F}, \mathcal{R}) \geq \epsilon$. The following theorem holds for a 4C forest scheme that uses a MAC scheme to compute the tags as described above (see Appendix D for the proof).

Theorem 2 *Let E be an adversary that $[t, q_s, q_v, \epsilon]$-breaks a MAC based 4C forest scheme, let $q = L(q_s + q_v + 1)$, where L is the maximum number of internal nodes that can appear in a forest, let $p_R = L \cdot 2^{-n_T} + 1 - \prod_{i=1}^{q}(1 - \frac{i}{2^{n_T}})$, and let $\epsilon > p_R$. Then, there is a statistical test U that $[t + cq, q, \frac{1}{2}(\epsilon - p_R)]$-breaks the finite function family $\mathcal{F}$ defined by the signing algorithm of the underlying MAC scheme, where $c > 0$ is a small implementation dependent constant.*

5.4 Security of ReMAC, MACC and SN-MAC

The security of SN-MAC, ReMAC and MACC follows trivially from the previous results. See Appendix F for more details.

References

[1] R. Anderson, F. Bergadano, B. Crispo, J. Lee, C. Manifavas, and R. Needham,“A New Family of Authentication Protocols,” ACM Operating Systems Review 32(4), pp. 9-20, 1998.

[2] M. Bellare, J. Killian, P. Rogaway,“The Security of Cipher Block Chaining Message Authentication Code,” Advances in Cryptology - Proceedings of Crypto’94, Lecture Notes in Computer Science, Vol. 839, pp. 341-358, Springer-Verlag, 1994.

[3] M. Bellare, R. Guerin and P. Rogaway, “XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions,” Advances in Cryptology - Proceedings of Crypto’95, Lecture Notes in Computer Science, Vol. 963, pp. 15-29, Springer-Verlag, 1995.

[4] F. Bergadano, D. Cavagnino, B. Crispo, “Chained Stream Authentication,” Proceeding of Selected Areas in Cryptography 2000, pp. 142-155, 2000.

[5] A. Berendschot, B. den Boer, J. P. Boly, A. Bosselaers, J. Brandt, D. Chaum, I. Damgard, M. Dichtl, W. Fumy, M. van der Ham, C. J. A. Jansen, P. Landrock, B. Preneel, G. Roelofsen, P. de Rooij, and J. Vandewalle, “Final Report of RACE Integrity Primitives,” LNCS 1007, Springer-Verlag, 1995.

[6] J. Black and P. Rogaway, “CBC MACs for Arbitrary-Length Messages: The Three-Key Construction,” Advances in Cryptology - Proceedings of Crypto 2000, Lecture Notes in Computer Science, vol. 1880, pp. 197-215, Springer-Verlag, 2000.

[7] J. Black and P. Rogaway, “A Block Cipher Mode of Operation for Parallelizable Message Authentication,” Advances in Cryptology - Proceedings of Eurocrypt 2002, Lecture Notes in Computer Science, vol. 2332, pp. 384-397, Springer-Verlag, 2002.

[8] D. Bleichenbacher and U. Maurer,“On the Efficiency of One-Time Digital Signatures,” Advances in Cryptology - Proceedings of AsiaCrypt ’96, LNCS 1163, Springer-Verlag, 1996.

[9] M. Burrows, M. Abadi and R. Needham,“A Logic of Authentication,” DEC SRC Research Report 39.

[10] S. Cheung,“An Efficient Message Authentication Scheme for Link State Routing,” Proceedings of the 13th Annual Computer Security Application Conference, 1997.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, “Introduction to Algorithms,” Second edition, McGraw-Hill, 2001.

[12] N. Ferguson and B. Schneier, “A Cryptographic Evaluation of IPsec,” technical report, 2000.

[13] FIPS PUB 198, The Keyed-Hash Message Authentication Code (HMAC).

[14] FIPS PUB 180-2, Secure Hash Standard.

[15] R. Gennaro and P. Rohatgi,“How to Sign Digital Streams,” Advances in Cryptology - Proceedings of Crypto ’97, LNCS 1294, Springer-Verlag, 1997, pp. 180-197.

[16] O. Goldreich, S. Goldwasser and S. Micali,“How to Construct Random Functions,” Journal of the ACM, Vol.33, No.4, 210-217, 1986.

[17] S. Goldwasser, S. Micali, and R. Rivest, “A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks,” SIAM Journal on Computing, 17(2):281-308, April 1988.

[18] R. Impagliazzo, L. Levin and M. Luby,"Pseudo-Random Generation from One-Way Functions," Proceedings of the Twenty First Annual Symposium on the Theory of Computing, ACM, 1989.

[19] T. Iwata and K. Kurosawa,"OMAC: One-Key CBC MAC," Fast Software Encryption, FSE 2003, LNCS 2887,pp.129-153, Springer,2003.

[20] E. Jaulmes, A. Joux and F. Valette,"On the Security of Randomized CBC-MAC Beyond the Birthday Paradox Limit: A New Construction," Fast Software Encryption, FSE 2002, LNCS 2365, pp. 237-251, Springer-Verlag.

[21] A. Joux, G. Martinet, and F. Valette, "Blockwise-Adaptive Attackers Revisiting the (In)Security of Some Provably Secure Encryption Modes: CBC, GEM, IACBC," Advances in Cryptology - Proceedings of Crypto 2002, Lecture Notes in Computer Science, vol. 2442, pp. 17-30, Springer-Verlag, 2002.

[22] C. Kaufman, R. Perlman and M. Speciner,"Network Security: Private Communication in a Public World," Prentice Hall, 2002.

[23] L. Lamport,"Constructing Digital Signatures From a One-Way Function," no. CSL 98, 1979.

[24] R. C. Merkle, "A Digital Signature Based on a Conventional Encryption Function," Advances in Cryptology - Proceedings of Crypto '87, LNCS 293, Springer-Verlag, 1987, pp. 369-378.

[25] S. Miner and J. Staddon, "Graph-Based Authentication of Digital Streams," The 2001 IEEE Symposium on Security and Privacy.

[26] C.Mitchell and M.Walker,"Solutions to the Multidestination Secure Electronic Mail Problem," Computers and Security, Vol.7(1988), pp.483-488.

[27] A. Perrig, R. Canneti, J. D. Tygar, D. Song, "Efficient Authentication and Signing of Multicast Streams Over Lossy Channels," Proceedings of the IEEE Security and Privacy Symposium, 2000.

[28] E. Petrank and C. Rackoff,"CBC MAC for Real-Time Data Sources," J. Cryptology, vol.13, no.3, pp.315-338, Springer-Verlag, 2000.

[29] A. D. Rubin and P. Honeyman,"Formal Methods for the Analysis of Authentication Protocols," CITI Technical Report 93-7, 1993.

[30] C. Schnor,"Efficient Identification and Signatures for Smart Cards," Advances in Cryptology - Crypto '89 Proceedings, LNCS 435, Springer-Verlag, 1989.

[31] C. K. Wong, S. S. Lam, "Digital Signatures for Flaws and Multicasts," Proceedings of IEEE ICNP '98, 1998.

[32] K. Zhang,"Efficient Protocols for Signing Routing Messages," Proceedings of the Symposium on Network and Distributed System Security, 1998.

A Proof of Lemma 2

We will first establish some relations between the ability of constructing partial forgery trees in 4C tree schemes and the ability of constructing forgeries and collisions for the underlying tag computation procedures. The attribute 4C will be omitted in this section. For example, when we say partial forgery tree, we mean partial forgery tree in a 4C scheme. Let's start with the case when the partial forgery tree is rooted at a TPS node. The result is trivial but we need to state it formally for later reference.

Lemma 3 *If T is a partial forgery tree rooted at a TPS node x, then the pair $(m(x), \tau(x))$ is a forgery pair.*

Proof. Assume that the pair $(m(x), \tau(x))$ is not a forgery pair. Then, the tag $\tau(x)$ was already computed by applying the procedure A^{TPS} on input $m(x)$ during some previous signing query. From the definition of partial forgery tree and the fact that $m(x)$ was already signed during some previous query it follows that T is not a partial forgery tree. ■

When the partial forgery tree is rooted at an FTT node x, the tag associated with x does not have to be forged. It is possible to construct partial forgery by finding collisions. The following lemma provides result about the collision properties of the sets of authentic pairs corresponding to two distinct valid trees whose roots have equal tags.

Lemma 4 *Let T_1 and T_2 be two distinct valid trees with FTT roots u_0 and v_0 correspondingly such that $\tau(u_0) = \tau(v_0)$ and $m(u_0) \neq m(v_0)$. Then, there are a pair $\mathbf{p_1}$ in the set of authentic pairs corresponding to T_1 and a pair $\mathbf{p_2}$ in the set of authentic pairs corresponding to T_2 such that $\mathbf{p_1}$ and $\mathbf{p_2}$ collide.*

Proof. Let us consider the recursive procedure **collision(T_1,T_2)**:

1. If the roots u_0 and v_0 have different number of children, then return 1.
2. If the tag of some child of u_0 is different from the tag of the corresponding child of v_0, then return 1.
3. If the color of some child of u_0 is different from the color of the corresponding child of v_0, then return 1.
4. If there is a child x of u_0 and a child y of v_0 such that x and y are both TPS or both degenerated and $m(x) \neq m(y)$, return 1.
5. If there are two distinct subtrees T_1' (rooted at u_1, a child of u_0) and T_2' (rooted at v_1, the corresponding child of v_0) such that $\tau(u_1) = \tau(v_1)$, $m(u_1) \neq m(v_1)$, and u_1 and v_1 are either both ITT or both FTT, then return **collision(T_1',T_2')**. Otherwise, return 0.

First, we will prove that the procedure always returns 1 when the trees T_1 and T_2 satisfy the conditions of the lemma. This is done in two steps: by proving that the procedure can not return 0, and by proving that the procedure can not run forever. Next, we show that a return value of 1 implies an existence of collision pairs.

Let $u_0, u_1, \ldots$ and $v_0, v_1, \ldots$ be the sequences of roots of the trees in the recursive procedure calls. The procedure will return zero only if there is some t so that none of the collision conditions (steps 1-4) is satisfied and there is no node u_t, a child of u_{t-1}, and a corresponding node v_t, a child of v_{t-1}, with the properties required in the last step of the procedure.

We will now show that, when T_1 and T_2 satisfy the assumptions of the lemma and none of the collision conditions is satisfied, the subtrees T_1' and T_2' in step 5 can always be constructed. If none of the collision conditions is satisfied, then u_0 and v_0 must have same number of nodes, and the tag and the color of any child of u_0 are equal to the tag and the color of the corresponding child of v_0. Since the message labels of u_0 and v_0 are different, there is a node u_1, a child of u_0, such that $m(u_1) \neq m(v_1)$, where v_1 is the child of v_0 that corresponds to u_1. Note that the colors and the tags of u_1 and v_1 must be same. The nodes u_1 and v_1 can not be degenerated or TPS because we assumed that the collision condition 4 is not satisfied. Hence, the nodes u_1 and v_1 are either both colored ITT or both colored FTT.

The analysis above can be extend to arbitrary u_t and v_t in a straightforward manner. Let u_{t-1} and v_{t-1} satisfy the requirements of step 5 ($\tau(u_1) = \tau(v_1)$, $m(u_1) \neq m(v_1)$, u_1 and v_1 are either both ITT or both FTT). Again, if the collision conditions are not satisfied for the trees rooted at u_{t-1} and v_{t-1}, then u_{t-1} and v_{t-1} must have same number of nodes, and the tag and the color of any child of u_{t-1} are equal to the tag and the color of the corresponding child of v_{t-1}. Since the message labels of u_{t-1} and v_{t-1} are different, there is a node u_t (a child of u_{t-1}) and a node v_t (the corresponding child of v_{t-1}) such that $m(u_t) \neq m(v_t)$. Note that the colors and the tags of u_t and v_t must be same. The nodes u_t and v_t can not be degenerated or TPS because we assumed that the collision condition 4 is not satisfied. Hence, the nodes u_t and v_t are either both colored ITT or both colored FTT. Hence, if the collision conditions are not satisfied, we can always find two subtrees that satisfy the requirements for recursive call in the fifth step. Furthermore, the depth of the node u_t is t and it can not be larger than the height of the tree. Hence, the procedure will finish in finite number of steps and it will return a value. According to the previous discussion, that value cannot be 0.

It is not difficult to see that if one of the collision condition is satisfied, then we can construct collision pairs. Since the procedure returns 1 if and only if one of the collision conditions is satisfied, it follows that for any valid trees that satisfy the assumptions of the lemma one can find collision pairs in the sets of authentic pairs corresponding to these trees. ■

The following lemma gives a relation between the set of authentic pairs corresponding to some 4C partial forgery tree and the set of known authentic pairs. More precisely, the set of authentic pairs corresponding to a partial forgery tree contains a forgery pair or a pair that collides with some of the known authentic pairs.

Lemma 5 *Let T be a partial forgery tree with a root x. At least one of the following statements is true:*

1. *$\tau(x)$ is forged.*
2. *There is a pair in the sets of authentic pairs corresponding to T that collides with some known authentic pair.*

Proof. If the root x of the partial forgery tree T is colored TPS, then by Lemma 3, the tag $\tau(x)$ is forged.

Let us consider the case when the root is FTT. Let $v_1, \ldots, v_k$ be the children of x. Assume that T is a partial forgery tree and the pair $((\tau(v_1)|| \ldots ||\tau(v_k)), \tau(x))$ is not a forgery pair, but a known authentic pair. Then, there is a valid tree T' generated as a result of some previous signing query such that $\tau(x) = \tau(x')$, where the FTT node x' is the root of T'. The message labels of the nodes x and x' must be distinct. Otherwise, the tree T is not a partial forgery tree. According to Lemma 4, there is a pair in the set of authentic pairs corresponding to T that collides with some pair in the set of authentic pairs corresponding to T' which is a subset of the set of all known authentic pairs. Hence, at least one of the statements must be true. ■

The following relation between partial forgery forests and the partial forgery trees will help us to extend the previous result to the case of 4C forest schemes.

Lemma 6 *Let x be a final node of some partial forgery forest consisting of the trees $T_1, \ldots, T_k$. At least one of trees $T_1, \ldots, T_k$ is a partial forgery tree for the underlying 4C tree scheme.*

Proof. First, we will consider the case when there is no known stream signature with the same stream number as the partial forgery forest. Then, the message label of the final node x is unique. Namely, the message $m(x)$ begins with the stream number and no message that begins with that number has been signed by the underlying 4C tree scheme. Therefore, the tree T_k is a partial forgery tree for the underlying tree scheme.

Now, assume that there is some previous signing query (there can be only one such query) whose stream number is same as the stream number of the partial forgery forest $T_1, \ldots, T_k$. Let $T'_1, \ldots, T'_n$ be the trees of the forest corresponding to that signing query. If the number of trees n is smaller than k, then again the message label $m(x)$ is unique (has unique tree number) and the tree T_k is a partial forgery tree for the underlying tree scheme. Finally, assume that $n \geq k$ and the tree rooted at x is not a partial forgery tree for the underlying tree scheme. This means that there is some node y in T'_k with the same message label as x. In that case, the forest $T_1, \ldots, T_k$ can be a partial forgery forest only if $k > 1$ and $M_1 || \ldots || M_{k-1} \neq M'_1 || \ldots || M'_{k-1}$, where M_i denotes the part of the stream associated with the tree T_i (derived from the message label of the root of T_i by discarding the prefix consisting of the stream number and the tree number) and M'_i denotes the part of the stream associated with the tree T'_i. Hence, there is an index $i < k$ such that M_i and M'_i are different. Since M_i and M'_i are different, the message labels $m(x_i)$ and $m(x'_i)$, where x_i is the root of T_i and x'_i is the root of T'_i, must be different also. Therefore, the stream $m(x_i)$ has never be signed by the underlying tree scheme and the tree T_i is a partial forgery tree for the underlying tree scheme. ■

The forgery or collision property (Lemma 2) follows from Lemma 5 and Lemma 6.

B Three colors for the intermediate nodes are not sufficient - Example

In this section, we present an example of a structure construction procedure that has the following property: If it is used in combination with a coloring procedure that uses less than four colors for the intermediate nodes, then the resulting stream authentication scheme can be broken regardless of the tag computation procedures that are in use.

Fig 8 depicts the possible structures of the stream authentication trees when the stream consists of one (Fig 8.a), two (Fig 8.b) or three (Fig 8.c) chunks. When the stream consists only of one chunk M_1, the chunk is divided into two parts M_{11} and M_{12} that are assigned as labels to the message nodes. A tag for each part of the chunk is computed at the degenerated nodes 2 and 3. The tags $\tau(2)$ and $\tau(3)$ are then used to compute the final tag $\tau(1)$ at the FTT node. If the

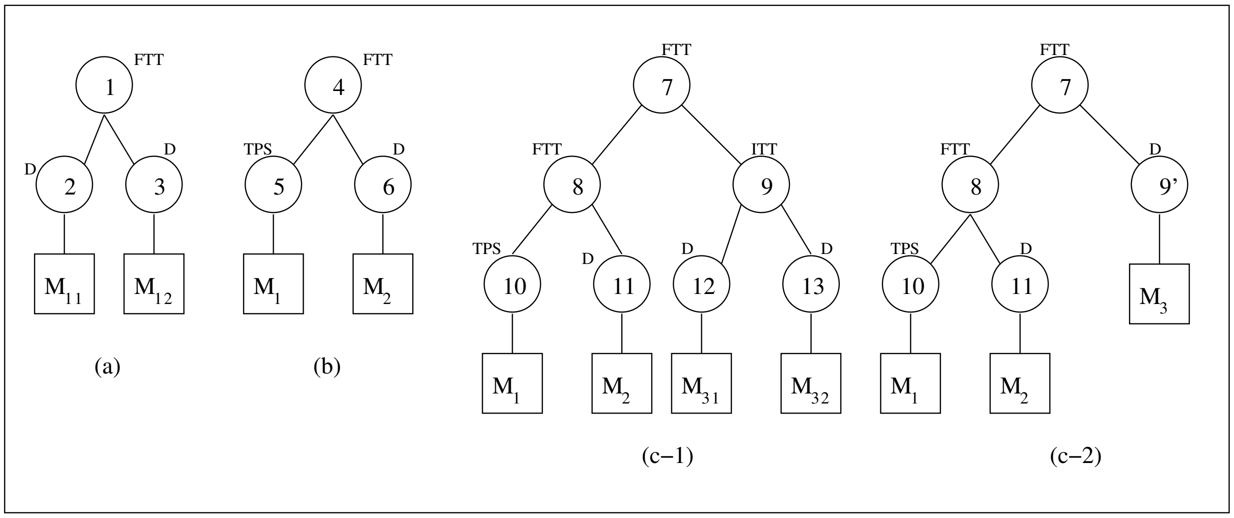


Figure 8: Possible structures when the stream consists of one, two or three chunks

stream consists of two chunks, then the first chunk is used to compute the tag $\tau(5)$ at the TPS node. The tag $\tau(5)$ is a temporary signature for the partial stream M_1. The second chunk M_2 is used to compute the tag $\tau(6)$. The final tag $\tau(4)$ is computed from the tags $\tau(5)$ and $\tau(6)$. When the stream consists of three chunks, the structure construction procedure flips a coin. Depending on the outcome, the procedure either divides the last chunks into two chunks M_{31} and M_{32}, and constructs the structure depicted in Fig 8.c.1, or constructs the structure depicted in Fig 8.c.2.

We will show that any coloring of these structures that uses at most three colors for the internal nodes is not secure. Assume that there is a secure stream authentication scheme that uses the previously described structure construction procedure and only three colors. If that is the case, then the most probable colors of the nodes 4, 5 and 6 must be different. Assume that the most probable colors of the nodes 5 and 6 are equal. Then, the adversary can submit a stream $(M_1, M_2), (M_1 \neq M_2)$ for signing and construct a partial forgery tree as follows. Take the subtree rooted at the node 6 and change the type of the node 6 from degenerated to TPS. The probability of the attack is $\geq \frac{1}{3}$ (since the most probable color appears with probability $\geq \frac{1}{3}$). This is in contradiction with our assumption that the scheme is secure. Assume now that the most probable colors of the nodes 4 and 5 are equal. The adversary can construct a partial forgery tree by discarding the message nodes, merging the nodes 5 and 6 into one message node whose label is $\tau(5)||\tau(6)$, and changing the type of 4 from FTT to TPS. The probability of success is large since the probability that $\tau(5)||\tau(6)$ will be equal to M_1 or $M_1||M_2$ for randomly selected M_1 and M_2 is very small. Otherwise, we will be able to break the scheme by guessing a node 5 tag. Again, this is in contradiction with our assumption that the scheme is secure, and thus, the most probable color of node 4 is different than the most probable color of 5. Finally, assume that the most probable color of 4 is equal to the most probable color of 6. Then we can use the degenerated node 6 to forge a node 4 tag. One possible construction of a partial forgery tree is depicted in Fig 9. Clearly, the success probability of the attack is high ($\geq \frac{1}{3}$), and, under our assumption that the scheme is secure, the most probable colors of 4 and 6 must be different.

An analogous analysis shows that all degenerated nodes must have same most probable color, and their most probable color must be different from the most probable colors of the TPS and FTT

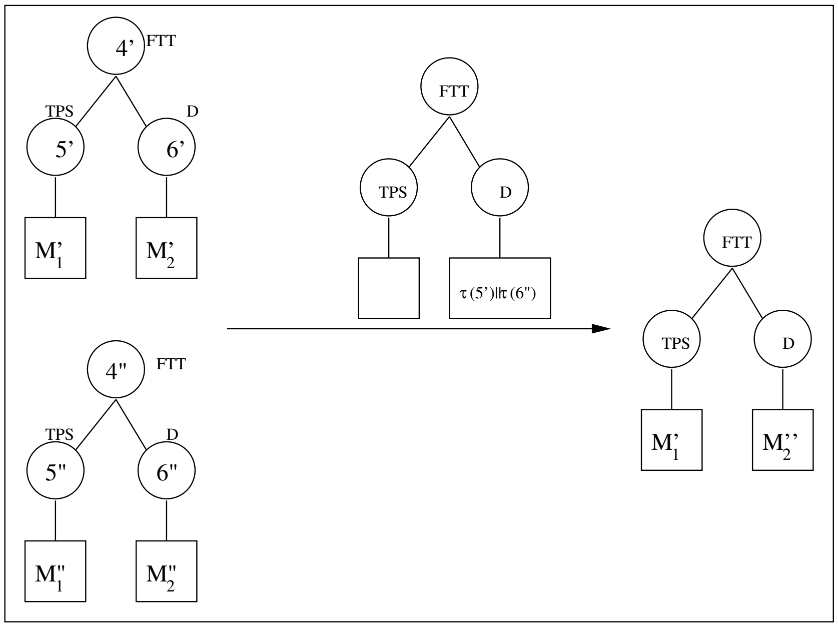


Figure 9: Constructing a forgery when 4 and 6 have same color

nodes. Similarly, all TPS nodes have same most probable color and it is different than the most probable color of the FTT and degenerated nodes. Now, we will show that if the most probable color of the ITT node 9 is equal to the FTT (resp., degenerated or TPS) most probable color, then we can break the scheme. Hence, there is no secure 3-coloring for the structure construction procedure described above. If the most probable color of 9 is equal to the FTT most probable color, then we can construct a partial forgery tree by taking the subtree rooted at 9 and changing the type of 9 from ITT to FTT. The derived stream authentication tree will will be of the type depicted in Fig 8.a. If the most probable color of the node 9 is equal to the TPS most probable color, then the adversary can construct a partial forgery in the following manner. Take the subtree rooted at 9. Discard the message nodes. Merge the degenerated nodes 12 and 13 into a message node whose label is $\tau(12)||\tau(13)$. Change the tag type of 9 from ITT to TPS. Finally, if the most probable color of the node 9 equals the degenerated most probable color, then the adversary can construct a partial forgery tree by converting the structure c-1 into the structure c-2 as follows. Discard the message nodes whose labels are M_{31} and M_{32}. Merge the degenerated nodes 12 and 13 into one message node whose label is $\tau(12)||\tau(13)$. Change the type of 9 from ITT to degenerated. If M_3 is different than $\tau(12)||\tau(13)$, then the constructed tree is partial forgery tree. The probability of the attack is small only if the chunk M_3 is equal to $\tau(12)||\tau(13)$ with large probability. However, in this case, we can break the scheme using different attack. It is not hard to see that if M_3 equals $\tau(12)||\tau(13)$ with high probability, then most of the time, the tags computed at the degenerated nodes are equal to the input used to compute them. In this case, we can mount an attack similar to the one depicted in Fig 9. Namely, we can assign tags computed at nodes 5 and 6 as message labels to the children of 2 and 3, and forge a node 4 FTT tag. ■

C Proof of Theorem 1

Suppose there is an adversary E that $[t', q'_s, q'_v, \epsilon]$-breaks a 4C forest scheme, and suppose that the underlying MAC is $[t'', q''_s, q''_v]$-forgeable on collision. We will construct an algorithm U that $[t' + t'' + cq, q, \epsilon]$-breaks the MAC scheme.

U's procedure is to run E and answer its oracle queries. In order to construct the answer to E's signing or verification query, U makes at most L queries to its oracle $\mathcal{O}$. U stores the result of each query in the memory[3] and checks for collision in a following manner. Let m be the message corresponding to the query, let τ be the tag corresponding to the query and let (m, τ) be valid. If there is no message stored at address $\tau_0 + \tau$, where τ_0 is some constant integer, U stores m at that location. If there is a message stored at address $\tau_0 + \tau$ and the message is equal to m, U doesn't store anything and continues. If there is a message stored at address $\tau_0 + \tau$ and the message is not equal to m, U invokes the algorithm A that constructs a forgery on collision. The A's input is the set of all pairs (m, τ) that are known authentic so far and the two pairs that collide. If no collision occurs when constructing the answers to E's queries, then E will finish and output its result. For each message/authenticator pair corresponding to the E's answer, U checks whether it is a forgery pair or it collides with some known authentic pair stored in the memory. If it is a forgery pair, U outputs it and halts. If it collides with some known authentic pair, then U invokes the algorithm A that produces a forgery on collision. Otherwise, U just halts.

The number of queries that U submits to its oracle $\mathcal{O}$ is not greater than $q = L(q'_s + q'_v + 1) + q''_s + q''_v$. The time complexity of U is at most $t' + t'' + cq$, where t' is the time required to break the stream authentication scheme, t'' is the time required to find a forgery pair for the MAC when MAC collision occurred, and cq is the time required to construct the answers to the queries. The stream authentication scheme is breakable with at least ϵ probability and the probability of forging a MAC when the stream scheme is broken is 1. Hence, the probability of breaking the MAC is at least ϵ. ■

D Proof of Theorem 2

Given an adversary E that $[t, q_s, q_v, \epsilon]$-breaks a MAC based 4C forest scheme, we will construct the statistical test U in a following manner.

U's procedure is to run E and answer its oracle queries. In order to construct the answer to E's signing or verification query, U makes at most L queries to its oracle $\mathcal{O}$. U stores the result of each query in the memory and checks for collision in a following manner. Let m be the message corresponding to the query, let τ be the tag corresponding to the query and let (m, τ) be valid. If there is no message stored at address $\tau_0 + \tau$, where τ_0 is some constant integer, U stores m at that location. If there is a message stored at address $\tau_0 + \tau$ and the message is equal to m, U doesn't store anything and continues. If there is a message stored at address $\tau_0 + \tau$ and the message is not equal to m, U outputs 1 indicating that collision occurred and halts. If there are no two pairs that collide in the pool of known authentic pairs, E will finish and output its result. For each message/authenticator pair corresponding to the E's answer, U checks whether it is a forgery pair or it collides with some known authentic pair stored in the memory. If so, it outputs 1 and halts. Otherwise, it outputs 0 and halts. The number of queries that U submits to its oracle $\mathcal{O}$ is at most $L(q_s + q_v) + L = q$, and the time complexity of U is at most $t + cq$, where $c > 0$ is a small

[3]We use RAM computational model.

implementation dependent constant.

Now, consider the following problem. Given a randomly selected function F from $\mathcal{R}$ and a pool P of q authentic pairs, output a pair that is a forgery or collides with some pair in q, when the function F is used for authentication. Let (m, τ) be the output of an algorithm A that solves the problem. If (m, τ) is not in P and it is a valid pair, then the algorithm is successful. Since F is randomly selected from $\mathcal{R}$ the probability of success in this case is at most $p_f = 2^{-n_T}$ for any algorithm. Consider now the case when the algorithm outputs a pair (m, τ) that is in P. The probability of success is at most $p_c = 1 - \prod_{i=1}^{q}(1 - \frac{i}{2^{n_T}})$, where $\prod_{i=1}^{q}(1 - \frac{i}{2^{n_T}})$ is the probability that there are no two pairs in P that collide. Hence, the probability that U will output 1, when the oracle is selected to be a random function from R is at most $p_R = L \cdot 2^{-n_T} + 1 - \prod_{i=1}^{q}(1 - \frac{i}{2^{n_T}})$. Therefore, the statistical test U that $[t + cq, q, \frac{1}{2}(\epsilon - p_R)]$-breaks the finite function family $\mathcal{F}$ defined by the signing algorithm of the underlying MAC scheme. ■

E Proof sketch for Proposition 1

Let $m_1 = m_1[1]||\ldots||m_1[n_1 - 1]||m_1[n_1]$ and $m_2 = m_2[1]||\ldots||m_2[n_2 - 1]||m_2[n_2]$ be two messages whose authenticators are identical when computed by the signing algorithm of the PMAC (resp., OMAC) scheme. The blocks $m_1[1], \ldots, m_1[n_1 - 1]$ of the message m_1 and the blocks $m_2[1], \ldots, m_2[n_2 - 1]$ of the message m_2 are of length l_B (the block length). The last blocks $m_1[n_1]$ and $m_2[n_2]$ can have length less than l_B and the length of $m_1[n_1]$ can be different from the length of $m_2[n_2]$.

By considering various cases for the possible lengths of $m_1[n_1]$ and $m_2[n_2]$, one can show that we can always compute $m_1' = m_1[1]||\ldots||m_1[n_1 - 1]||m_1'[n_1]$ and $m_2' = m_2[1]||\ldots||m_2[n_2 - 1]||m_2'[n_2]$ so that:

1. $m_1' \neq m_2'$
2. $\{m_1', m_2'\} \neq \{m_1, m_2\}$.
3. $\text{pad}(m_1'[n_1]) \oplus \text{pad}(m_2'[n_2]) = \text{pad}(m_1[n_1]) \oplus \text{pad}(m_2[n_2])$
4. The length of $m_1'[n_1]$ (resp., $m_2'[n_2]$) is less than l_B if and only if the length of $m_1[n_1]$ (resp., $m_2[n_2]$) is less than l_B.

where **pad** is the function that is used to pad the last block to length l_B. It is not hard to verify that two messages that satisfy the conditions above have identical authenticators when PMAC (resp., OMAC) is used for signing.

An algorithm that will forge on first collision works as follows. Given two message/authenticator pairs (m_1, τ) and (m_2, τ), we compute the messages m_1' and m_2'. Then, we seek in the set of known authentic pairs whether there is already a computed authenticator for m_1' or m_2'. If there is an already computed authenticator τ' for the message m_1' (resp., m_2'), then the algorithm outputs (m_2', τ') (resp., (m_1', τ')). The pair (m_2', τ') (resp., (m_1', τ')) is a forgery since (m_1, τ) and (m_2, τ) are the only pairs in P that collide. If there is no already computed authenticator for one of the messages m_1' and m_2', then the algorithm submits m_1' for signing and outputs (m_2', τ'), where τ' is the answer to the signing query. ■

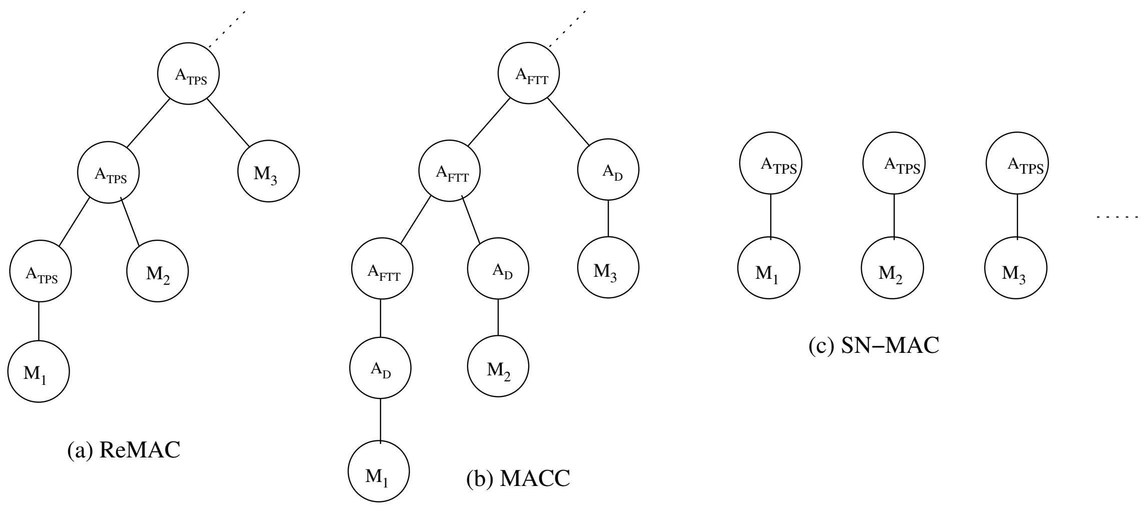


Figure 10: Practical stream authentication schemes

F Security of ReMAC, MACC and SN-MAC

A 4C forest representation of the schemes presented in Section 3 is given in Fig 10. ReMAC can be viewed as a 4C forest scheme where the forest consists only of one tree and all internal nodes are TPS nodes (see Fig 10.a). The following corollary establishes a relation between the security of a ReMAC scheme and the security of the underlying MAC scheme.

Corollary 1 *If there is an adversary that $[t, q_s, q_v, \epsilon]$-breaks ReMAC, then there is an adversary that $[t + cL(q_s + q_v + 1), L(q_s + q_v + 1), \epsilon]$-breaks the underlying MAC scheme, where $c > 0$ is a small implementation dependent constant and L is the maximum number of chunks that can appear in a stream.*

The corollary follows from the Theorem 2 and Lemma 3. Since all internal nodes in the forest are TPS nodes, the adversary will always be able to find a forgery for the underlying scheme.

MACC scheme is a 4C tree scheme (or equivalently a forest scheme where the forest always has only one tree) such that there are no ITT or TPS nodes (see Fig 10.b). The security of the scheme follows from Theorem 2.

Corollary 2 *If there is an adversary that $[t, q_s, q_v, \epsilon]$-breaks the MACC scheme, then there is a statistical test that $[t + cq, q, \frac{1}{2}(\epsilon - p_R)]$-breaks the finite function family $\mathcal{F}$ defined by the signing algorithm of the underlying MAC scheme, where $c > 0$ is a small implementation dependent constant, L is twice the maximum number of chunks that can appear in a stream, $q = L(q_s + q_v + 1)$ and $p_R = L \cdot 2^{-n_T} + 1 - \prod_{i=1}^{q}(1 - \frac{i}{2^{n_T}})$.*

SN-MAC corresponds to a 4C forest scheme where the trees consist of only two nodes: a TPS root and a leaf (see Fig 10.c). Theorem 2 and Lemma 3 imply the following corollary about the security of SN-MAC.

Corollary 3 *If there is an adversary that $[t, q_s, q_v, \epsilon]$-breaks SN-MAC, then there is an adversary that $[t + cL(q_s + q_v + 1), L(q_s + q_v + 1), \epsilon]$-breaks the underlying MAC scheme, where $c > 0$ is a small* *implementation dependent constant and L is the maximum number of chunks that can appear in a stream.*