Chained Stream Authentication*

Francesco Bergadano[1], Davide Cavagnino[1], and Bruno Crispo[2,**]

[1] Computer Science Department, University of Turin,
Corso Svizzera 185, 10149 Torino, Italy
{bergadan,davide}@di.unito.it
[2] SRI International, Information Assurance Lab Cambridge,
23 Millers Yard, Mill Lane, Cambridge CB2 1RQ, UK
crispo@cam.sri.com

Abstract. We present a protocol for the exchange of individually authenticated data streams among N parties. Our authentication procedure is fast, because it only requires the computation of hash functions – we do not need digital signatures, that are substantially less efficient. The authentication information is also short: two hash values for every block of data. Since there are no shared secrets, this information does not grow with N, the number of parties.

1 Introduction

Multicast applications are receiving increasing commercial interest. In particular, multicast conferencing tools are available that support real-time multiway communications over a packet network.

The security of multicast and videoconferencing has also become an important and specific issue [8,3,21], and it includes two forms of authentication services:

- *Group authentication.* The conference participants share one key, and authentication allows to check that data has not been modified or inserted by attackers *outside* the group [21,24,3].
- *Individual authentication.* Received data streams must be authenticated with respect to their individual origin, i.e. one individual group member [8].

This paper addresses individual authentication. An authentication protocol will be obtained, that is secure even when attackers may adaptively choose authenticated data streams. For an interactive scenario, that is typical of multicast conferencing, the proposed solution is substantially more efficient than in previous approaches.

* The protocol described in this paper, and the timing protocol derived from it, were first presented in a seminar at IBM T. J. Watson Research Center in summer 1998 by F. Bergadano

** This work was completed while Bruno Crispo was with the Computer Science Department, University of Turin

D.R. Stinson and S. Tavares (Eds.): SAC 2000, LNCS 2012, pp. 144–157, 2001.

2 Previous Work

2.1 Individual Authentication in Multicast Groups

Individual authentication may be essential in multicast applications. Yet, it is difficult to achieve. If one uses digital signatures, the required computational cost may be too high. In fact, end-user hardware may be limited and loaded by the multimedia processing that is part of the conferencing application. If one uses symmetric Message Authentication Codes (MACs), every stream block will need to carry a distinguished code for every recipient. Previous work on individual authentication for multicast is all based on one of the above schemes (signature or multiple MACs), with modifications directed to improve efficiency.

On the side of symmetric authentication, it is possible to use a fixed number K of MAC keys, that does not depend on the size N of the multicast group [8,10]. The sender knows K keys, and each receiver knows K/2 keys, chosen at random. Each stream block is authenticated with K MACs, corresponding to the K keys, and receivers check the MACs for which they have a key. Obviously, receiver collusions can lead to forged individual authentication codes.

More work is available on the side of efficient digital signatures. Online/offline signatures [11] may be used to split the computation in a more expensive off-line phase, followed by an efficient online phase that is needed when the data becomes available. On the other hand, part of the signature can be performed by a "signature server", that may be located elsewhere, without compromising authentication or even non-repudiation [2]. One-time signatures [18,7] are an efficient alternative, and are adequate for stream authentication. Gennaro and Rohatgi [12] have proposed an efficient stream signing method that is based on a chain of one-time signatures. The disadvantage of this approach lies in the length of one-time signatures.

2.2 Hash Chains

Our work does not fall within the two above categories. In particular, it is not based on asymmetric cryptography, and there are no shared secrets. We use single individual MACs, and a hash chain of individual, secret keys. Hash chains have been used for a long time and for a number of different purposes.

Hash chains were first proposed by Lamport [17] and then used in the S/Key [15] user identification system. They have been applied to the authentication of public key certificate revocation/validity messages [20], to digital payment systems [23], and to Web server hit acknowledgements. All of the above applications basically use hash chains to send periodic "yes, I'm alive" messages in a secure way. In other words, they are *contentless*, meaning that content is bound to the chain at the start, but is not added or modified when the individual hash values are released.

This is not the case with more recent approaches, where hash chains have been proposed for the authentication of messages, as described in [2,1]. The basic idea of [1] is to MAC each block of data with a new key, and send the key as part

of the next message. The main problem with that technique is that when a key is sent too early (i.e. before the previous message was obtained by all intended recipients), falsifying the rest of the stream becomes possible.

A time protocol based on hash chains was developed independently by [9] and by [5]. This protocol was also published in [22] and [6].

In this paper we present an interactive protocol that does not depend on time, and provide proofs of its security.

3 Chained Stream Authentication

Following the formalization of [13], and [12], we define a *security parameter*, n, and say that a function $\epsilon(n)$ is "negligible", if, for all constants c, there is n_0 such that, for $n > n_0$, $\epsilon(n) < 1/n^c$.

Following [13], we define a signature scheme as a triple (G,Sig,V) of probabilistic polynomial time algorithms, where (1) G is used to generate a key pair (SK,PK), (2) Sig is used to sign any message M, using the secret key SK, and (3) V is the signature verification algorithm, such that V(PK,M,Sig(SK,M))=1, for any message M. We will use a signature scheme that is secure against adaptively chosen message attacks [13]: the probability of forging a signature is negligible, even when a signature oracle is available.

Similarly, we define a *stream authentication scheme* as a triple of probabilistic polynomial-time algorithms (GA,AA,VA), where

- On input 1^n, GA outputs a pair of keys (SK,PK)$\in \{0,1\}^{2n}$.
- AA is the authentication algorithm, and receives in input a secret key SK, and a stream $S = S_1, S_2, ..., S_i$, consisting of a finite number i of *blocks*. AA outputs an authenticated stream $S' = S'_1, ..., S'_i$, where $S'_j = (S_j, auth_j)$, being $auth_j$ some kind of authentication data.
- The verification algorithm VA is such that VA(PK,AA(SK,S))=1. When VA(PK,S')=1, we will say that S' is *valid*.

We may now define our proposed scheme, called a *chained stream authentication scheme*:

- As a generator GA, we use the generator of a signature scheme (G,Sig,V), secure against chosen message attacks.
- The authentication algorithm AA will be called a "Chained Stream Authentication" algorithm (CSA). This algorithm first generates a secret α, computes $h^k(\alpha)$ for some $k > i$, and then produces the following output:

$$\begin{aligned} S'_1 &= S_1,\ MAC_{h^{k-1}(\alpha)}(S_1), h^k(\alpha), SN, Sig(SK, h^k(\alpha), SN) \\ S'_2 &= S_2,\ MAC_{h^{k-2}(\alpha)}(S_2), h^{k-1}(\alpha) \\ &\quad \text{..... .....} \\ S'_i &= S_i,\ MAC_{h^{k-i}(\alpha)}(S_i), h^{k-i+1}(\alpha) \end{aligned}$$

  By MAC, we denote a secure Message Authentication Code, i.e. such that the probability of forging a valid code is negligible, even when a MAC oracle is available. For h, we will use a collision resistant hash function. The

authentication includes a session number SN that is incremented for every new stream.
- The verification algorithm VA will ouput 1 if the initial asymmetric signature is valid, if all the MACs are correct, and if the hash chain of the MAC keys is consistent, i.e. the hash of a key produces the previous key in the chain.

4 Security against Continuations

What are the security properties of the proposed scheme? Clearly, given a stream authentication oracle, it is possible to forge new valid authenticated streams, because the MAC keys become known. Therefore, CSA is not "secure" according to the definition of [12], that can be rephrased as follows: *"a stream authentication scheme (GA,AA,VA) is secure if any probabilistic polynomial-time algorithm F, given as input the public key PK and adaptively chosen authenticated streams $S'^{(j)}$, outputs a new valid authenticated stream $S' \not\subseteq S'^{(j)}$, for all j, only with negligible probability".* Clearly, this definition of security does not apply to the CSA scheme: the forger F may ask for just one authenticated stream, change any block but the last, and recompute the corresponding MAC using the available key.

However, our scheme satisfies a weaker security notion, which we will call "security against continuations". We define continuations as follows:

Definition. *A stream S_2 is a* **continuation** *of a stream S_1, denoted by $S_1 \subset S_2$, if S_1 is a proper prefix of S_2.*

The same definition applies to authenticated streams under (GA,AA,VA). A valid authenticated continuation S'_2 of an authenticated stream S'_1 must then be such that $S'_1 \subset S'_2$ and $VA(PK, S'_2) = 1$. Security against continuations means, informally, that it is unfeasible to produce valid continuations of observed valid streams. This weaker notion will nevertheless be sufficient for building secure authentication protocols, after some means of sender/receiver syncronization is achieved, as described in Sections 5 and 6. More precisely, security against continuations corresponds to the following:

Intuition. *A forger may produce a new valid stream S only if it is associated to a stream T, that was used previously. Moreover, if $|S| = |T|$, and the last blocks of S and T are different, then it will be impossible to produce a valid continuation of S.*

Next, we formalize the above intuition and prove that it applies to CSA (in Lemma 1, with proof given in Appendix A).

Definition. *A stream authentication scheme is* **secure against continuations** *if there is no polynomial time algorithm F that, given adaptively chosen authenticated streams $S'^{(1)}, ..., S'^{(k)}$, is able to generate a valid authenticated stream $S' = S'_1, ..., S'_j$ with non-negligible probability, unless $\exists i \in [1, k]$ such that $S'^{(i)}_1 = S^{(i)}_1$, MAC^i_1, H, SN, $sig(H, SN)$, where $S'_1 = S_1$, MAC_1, H, SN, $sig(H, SN)$, and one of the following holds:*

1. *$j < |S'^{(i)}|$, or*

2. $j = |S'^{(i)}|$, *and* $S'_j = S'^{(i)}_j$, *or*
3. $j = |S'^{(i)}|$, *and* $S'_j \neq S'^{(i)}_j$, *and there is no polynomial time algorithm F' that can generate with non-negligible probability a valid continuation* $T' \supset S'$, *given* $S'^{(1)}, ..., S'^{(k)}$, *possibly other adaptively chosen authenticated streams, and any valid authenticated continuation of* $S'^{(i)}$.

Security against continuations is a complicated notion, but it will lead us to a simple concept of stream authentication in Theorem 1. First, though, we need the following:

Lemma 1. *Suppose that, in the CSA authentication scheme,*
(1) (G,S,V) is a secure signature scheme,
(2) g is a pseudorandom function,
(3) $h(x) = g_x(0)$ *and* $MAC_k(x) = g_k(1, x)$
(4) g is such that h is a collision resistant hash function.
Then, the scheme (GA,CSA,VA) is secure against continuations.

MAC and h are of the CSA algorithm, and are defined through a pseudorandom function [14,4] as defined in (3) above, because not only should the MAC be secure, but each key k must look random even though $h(k)$ is known. With the definition of (3), knowing $h(k) = g_k(0)$ gives no additional information, as one could in any case query the oracle for g_k and obtain $g_k(0)$. In practice, one could use g=HMAC [16] so as to satisfy both (2) and (4).

5 The Chained Stream Authentication Protocol with One Sender and One Receiver

We will now use the CSA scheme to authenticate information over an insecure network. In fact, CSA's security against continuations can be used with a synchronization mechanism to obtain a very efficient individual authentication method. For now, we consider one party, named A, who will send authenticated data, and one party, named B, who will receive the data. The protocol is defined below, where $Sig_A(x) = Sig(SK_A, x)$ is A's signature under (G,S,V), and similarly $Sig_B(x) = Sig(SK_B, x)$ for B:

1. B $\rightarrow$ A: $h^k(\beta), SN, Sig_B(h^k(\beta), SN)$
   A $\rightarrow$ B: $A_1, MAC_{h^{k-1}(\alpha)}(A_1), h^k(\alpha), SN, Sig_A(h^k(\alpha), SN)$
2. B $\rightarrow$ A: $h^{k-1}(\beta)$
   A $\rightarrow$ B: $A_2, MAC_{h^{k-2}(\alpha)}(A_2), h^{k-1}(\alpha)$

...

i. B $\rightarrow$ A: $h^{k-i+1}(\beta)$
   A $\rightarrow$ B: $A_i, MAC_{h^{k-i}(\alpha)}(A_i), h^{k-i+1}(\alpha)$

...

Messages are sequential: A will not send message i if it has not received a correct i-th message from B, and B will not send message $i + 1$ if it has not received from A a correct i-th message. A and B initially generate individual

random secrets α and β, and compute $h^k(\alpha)$ and $h^k(\beta)$, respectively. These values, and the session number SN, are signed, and exchanged as part of the first messages in step 1. Then, A sends data as defined in the CSA scheme, and B sends back authenticated acknowledgments. B's authenticated ack for A's j-th message is simply $h^{k-j}(\beta)$. The receiver side is then similar to what happens in S/Key and similar applications [15,17]. The security properties of the protocol, to be discussed next, are made relative to the following:

Active Attack Model. CSA sender A, CSA receiver B, and attacker E:

- *E runs in polynomial time and may ask A to send B the authenticated streams* $S^{(1)}, ..., S^{(k)}$ *in sessions* $1, ..., k$*;*
- *A chooses stream* $S^{(k+1)}$ *and sends it to B in session* $k+1$*;*
- *At any time, E can read messages, stop messages and insert messages;*
- *During session* $k+1$*, E tries to have B receive* $S' \neq S^{(k+1)}$ *and believe it authentic.*

We call the above an *active stream authentication attack*. We shall prove in Theorem 1 that such attacks are not feasible with the above CSA protocol, except for the possible falsification of the last block of $S^{(k+1)}$. We first note that session numberings by sender and receiver are consistent:

Observation 1. *Suppose A has sent its first message of session SN. Then B must have already sent its first message of session SN.*

Observation 2. *Suppose B has sent its second message of session SN. Then A must have already sent its first message of session SN.*

The observations allow us to speak of a "current session" in the CSA protocol with one sender and one receiver. We can now prove that this protocol represents a valid authentication mechanism:

Theorem 1 (Security of CSA with one sender and one receiver). *Suppose sender A and receiver B run SN sessions under the CSA protocol, where:*

- *the conditions of Lemma 1 hold;*
- *a polynomial active attacker E chooses streams* $A^{(1)}, ..., A^{(SN-1)}$*, that A sends to B in sessions* $1, ..., SN-1$*;*
- *B has received the valid authenticated stream* $S' = S'_1, ..., S'_j$ *during session SN.*

Then, E can cause $S'_1, ..., S'_{j-1}$ *to be non-authentic only with negligible probability.*

The proof is by induction on $|S'|$, based on Lemma 1 (see Appendix A). The last block of S' may be modified by E. Thus, authentication is obtained with a one block delay: the receiver can ascertain the origin and the integrity of a block in the stream only after receiving the next block. This is acceptable in most multicast applications. This is achieved without shared secrets and without signatures after the first message. The important consequences of this fact are discussed in the next session, where the protocol is used with more than just two parties.

6 The N-Party CSA Protocol

We will now extend the protocol so that it can be used effectively in a multicast conferencing scenario. As a first step, we will consider two parties, A and B, that must exchange authenticated data in both directions. Obviously, this can be done by simply applying the CSA protocol with sender A and receiver B, and simultaneously with sender B and receiver A. However, we can make this more efficient by merging the hash values sent as acknowledgments and the ones sent as hashes of secrets. This results in the following **two-party protocol**:

1. B $\rightarrow$ A: $B_1, MAC_{h^{k-1}(\beta)}(B_1), h^k(\beta), SN, Sig_B(h^k(\beta), SN)$
   A $\rightarrow$ B: $A_1, MAC_{h^{k-1}(\alpha)}(A_1), h^k(\alpha), SN, Sig_A(h^k(\alpha), SN)$
2. B $\rightarrow$ A: $B_2, MAC_{h^{k-2}(\beta)}(B_2), h^{k-1}(\beta)$
   A $\rightarrow$ B: $A_2, MAC_{h^{k-2}(\alpha)}(A_2), h^{k-1}(\alpha)$

...

i. B $\rightarrow$ A: $B_i, MAC_{h^{k-i}(\beta)}(B_i), h^{k-i+1}(\beta)$
   A $\rightarrow$ B: $A_i, MAC_{h^{k-i}(\alpha)}(A_i), h^{k-i+1}(\alpha)$

...

The above protocol may cause practical transmission difficulties. In particular, each party may send a block of data only after receiving a corresponding block from the other party. This causes a kind of stop-and-wait behaviour that implies poor network utilization and may result in unacceptable delays for real-time traffic. Fortunately, such strict sequentialization of messages is not necessary. In particular, data blocks and MACs can be sent at any time - only the delivery of keys need be delayed until acknowledgements are obtained and verified. We may therefore rewrite the above two party protocol by splitting the behaviour of each party into a *data sender* process and a *key sender* process, that run independently. For party A, the processes are defined as follows (party B is defined symmetrically):
A's data sender process:

1. send to B: $MAC_{h^{k-1}(\alpha)}(A_1)$, A_1
2. send to B: $MAC_{h^{k-2}(\alpha)}(A_2)$, A_2

...

A's key sender process:

1. wait for $MAC_{h^{k-1}(\beta)}(B_1)$
   send to B: $h^k(\alpha), SN, sig_A(h^k(\alpha), SN)$
2. wait for $MAC_{h^{k-2}(\beta)}(B_2)$
   wait for $h^k(\beta), SN, sig_B(h^k(\beta), SN)$
   send to B: $h^{k-1}(\alpha)$
3. wait for $MAC_{h^{k-3}(\beta)}(B_3)$
   wait for $h^{k-1}(\beta)$
   send to B: $h^{k-2}(\alpha)$

...

Delaying just secrets, not information, is essential in multicast conferencing: the receiver will continue viewing the stream of data even though the keys necessary to authenticate it are not yet available. When secrets are late, viewing is ahead of authentication, and we call this an *authentication delay*. The delay would be small and roughly equivalent to three times the network latency. The reason is that a MAC must be sent, then the authenticated acknowledgement is returned, and finally the MAC key is sent. Only then can the corresponding block be authenticated by the receiver.

We may now generalize the above construction and obtain the **N-party protocol**. During session SN, party i first generates a random secret α_i and computes $h^k(\alpha_i)$. Party i consists of two processes, a *data sender* and a *key sender*, that run concurrently as described below, where $A_{i,j}$ is the jth block sent by party i:
Data sender i:

1. multicast $MAC_{h^{k-1}(\alpha_i)}(A_{i,1})$ and $A_{i,1}$;
2. multicast $MAC_{h^{k-2}(\alpha_i)}(A_{i,2})$ and $A_{i,2}$;

...

Key sender i:

1. wait for $MAC_{h^{k-1}(\alpha_j)}(A_{j,1})$, for all $j \in [1, N]$;
   multicast $h^k(\alpha_i), SN, i, sig_i(h^k(\alpha_i), SN, i)$;
2. wait for $MAC_{h^{k-2}(\alpha_j)}(A_{j,2})$, for all $j \in [1, N]$;
   wait for $h^k(\alpha_j), SN, j, sig_j(h^k(\alpha_j), SN, j)$, for all $j \in [1, N]$;
   multicast $h^{k-1}(\alpha_i)$;
3. wait for $MAC_{h^{k-3}(\alpha_j)}(A_{j,3})$, for all $j \in [1, N]$;
   wait for $h^{k-1}(\alpha_j)$, for all $j \in [1, N]$;
   multicast $h^{k-2}(\alpha_i)$;

...

It is important to note that, for every block, the only authentication information that is multicast is one MAC (sent by the data sender) and one hash value (sent by the key sender). This does not grow with N.

Conclusions

We have defined a Chained Stream Authentication scheme and proved it to be secure against non-authentic stream continuations. This property was the basis for an interactive stream authentication protocol that was proven to be secure against active attacks, even when the attacker may ask for the authentication of a number of adaptively chosen streams. The protocol was then optimized for the case of a bidirectional flow of data.

However, the importance of the protocol arises when there are more than just two communicating parties. Since there are no shared secrets, the size of the authentication data does not grow linearly with the number of parties. In

the optimized protocol for N parties, we need just one MAC and one hash value for every block of data. By contrast, symmetric individual authentication would require N distinguished MACs for every block. Therefore, the CSA protocol represents a major improvement. If compared with techniques based on the digital signature of each block, CSA is to be preferred because of its much higher efficiency.

With respect to the work in [12], our scheme has the advantage that if a packet is lost, using a timeout when waiting for all the MACs and keys, the authentication may proceed for the following blocks due to the use of a chain of keys, while in [12] a packet may be verified only if all the preceding packets have been received. Moreover, the online solution in [12] uses one-time signatures, that introduce a communication overhead of the order of 1000 bytes per packet. The CSA solution is an order of magnitude more efficient than [12] with respect to the authentication information transmitted by the sender. Another difference with our work is that [12] allows non-repudiation, while our scheme does not.

Appendix A (Proof of Lemma 1 and Theorem 1)

Proof of Lemma 1. Suppose that a forger F exists that can produce a valid authenticated stream S' with a non-negligible probability ϵ, contraddicting the thesis. Then, one of the two following cases must hold, and at least one must hold with probability at least $\epsilon/2$:

Case 1 $S'_1 = S_1, MAC_1, H, SN, sig(H, SN)$ and there is no $S'^{(i)}$ such that $S'^{(i)}_1 = S'^{(i)}_1, MAC^{(i)}_1, H, SN, sig(H, SN)$. Then, we can use the forger F to construct algorithm F1 that breaks the asymmetric signature scheme (G,S,V). The constructed algorithm F1 has access to an oracle for S, and starts by calling F as a subroutine. When F requires an authenticated stream $S'^{(i)}$, F1 generates a random secret $\alpha^{(i)}$, computes $h^k(\alpha^{(i)})$, and asks the oracle for $sig(h^k(\alpha^{(i)}), SN)$. Then, knowing $\alpha^{(i)}$, F1 authenticates the rest of the stream as required by CSA, and outputs the authenticated stream $S'^{(i)}$ for F. The process continues until, with probability at least $\epsilon/2$, F outputs the new valid authenticated stream S'. In this case (Case 1), S' must be such that $S'_1 = S_1, MAC_1, H, SN, sig(H, SN)$ and there is no $S'^{(i)}$ generated by F1 such that $S'^{(i)}_1 = S'^{(i)}_1, MAC^{(i)}_1, H, SN, sig(H, SN)$. This means that a signature $sig(H, SN)$ was never queried by F1 to the oracle. Hence F1 breaks (S,G,V) by outputting the new valid signature $H, SN, sig(H, SN)$.

Case 2 $S'_1 = S_1, MAC_1, H, SN, sig(H, SN)$ and there is $S'^{(i)}$ such that $S'^{(i)}_1 = S'^{(i)}_1, MAC^{(i)}_1, H, SN, sig(H, SN)$. Then there are three cases, and one must hold with probability at least $\epsilon/6$:

Case 2.1 $|S'| < |S'^{(i)}|$. This case does not contraddict the Lemma.

Case 2.2 $|S'| > |S'^{(i)}|$. In this case we can construct F2 that can invert h, using the forger F, and hence break g. The inverter F2 is given a value α and must compute $h^{-1}(\alpha)$ with non-negligible probability. F2 calls GA to generate a key pair (SK,PK), and then runs F as a subroutine. Let SN_{max} be the maximum

number of authenticated streams that F will ask for. Clearly SN_{max} must be polynomially large. F2 will then pick a random number R between 1 and SN_{max}. When asked to authenticate stream $S^{(R)}$, F2 lets $\alpha^{(R)} = \alpha$ and then follows the CSA authentication algorithm, but choses $k = |S^{(R)}|$. When F stops, it will output S' such that $|S'| > |S'^{(i)}|$, with probability greater than $\epsilon/6$, and $i = R$ with probability $1/SN_{max}$. Hence, with probability greater than $\epsilon/(6 * SN_{max})$, $S'_{|S'^{(i)}|+1} = (S, MAC, h^{k-(|S'^{(i)}|+1)}(\alpha)) = (S, MAC, h^{k-(k+1)}(\alpha)) = (S, MAC, h^{-1}(\alpha))$. Since $h(x) = g_x(0)$, inverting h means breaking g_{key} after observing $g_{key}(0)$.

Case 2.3 $|S'| = |S'^{(i)}| = j$. There are two cases, and one must hold with probability at least $\epsilon/12$:

Case 2.3.1 $S_j = S_j^{(i)}$. This case does not contraddict the Lemma.

Case 2.3.2 $S_j \neq S_j^{(i)}$. Let $S'_j = S_j, MAC_{key}(S_j), h^{k-j+1}(\alpha)$. There are two cases, and at least one must occur with probability at least $\epsilon/24$:

Case 2.3.2.1 $MAC_{key}(S_j)$, generated by F, and $MAC(S_j^{(i)})$, given in stream $S'^{(i)}$, are valid under the same key. In this case we can use F to break g_{uk}, for some unknown key uk, in a polynomial algorithm F3, that has access to an oracle for g_{uk}. Define again SN_{max} as the maximum number of authenticated streams that F will ask for. F3 will then pick a random number R between 1 and SN_{max}. F3 will run F as a subroutine, will use GA to generate a key pair (SK,PK), and will authenticate all streams requested by F normally using CSA, except for stream $S^{(R)}$. For this stream, define $l = |S^{(R)}|$, and let $\alpha^{(R)} = uk$, and $k = l$. Then, $h^{k-l+1}(\alpha^{(R)}) = h(uk)$. F3 then computes $h^{l-1}(uk), ..., h(uk)$, after querying the oracle for $h(uk) = g_{uk}(0)$, and uses these values, in this order, to compute the MACs for $S_1^{(R)}, ..., S_{l-1}^{(R)}$, as required in CSA. As the last authenticated block, F3 outputs $S_l'^{(R)} = (S_l^{(R)}, MAC_{uk}(S_l^{(R)}), h(uk))$, where $MAC_{uk}(S_l^{(R)}) = g_{uk}(1, S_l^{(R)})$, is queried to the oracle. With probability $1/SN_{max}$, the authenticated stream S' output by F is associated to stream $S^{(R)}$, i.e., $i = R$. In this case, $MAC_{uk}(S_j^{(R)})$ and $MAC_{key}(S_j)$, are valid under the same key, i.e., $key = uk$. F3 then ouputs $MAC_{uk}(S_j) = g_{uk}(1, S_j)$, and since $S_j \neq S_j^{(i)}$, this is a new forged MAC, and a correct value of g_{uk} for the new input $(1, S_j)$, that is generated with non-negligible probability greater than $\epsilon/(24 * SN_{max})$.

Case 2.3.2.2 $MAC_{key}(S_j)$, generated by F, and $MAC(S_j^{(i)})$, given in stream $S'^{(i)}$, are not valid under the same key. We show that, in this case, there is no polynomial time algorithm F' that can generate with non-negligible probability a valid authenticated continuation $T' \supset S'$, given $S'^{(1)}, ..., S'^{(k)}$, possibly other adaptively chosen authenticated streams $T'^{(1)}, ..., T'^{(p)}$, and any valid continuation of $S'^{(i)}$. Suppose such a forger F' exists, and does the above with non-negligible probability ϵ'. Let $T'_{j+1} = (T_{j+1}, MAC, key)$. Since T' is valid and is a continuation of S', we also know that $T'_j = S'_j = (S_j, MAC_{key}(S_j), h^{k-j+1}(\alpha))$. We construct F4 that can generate collisions for h with non-negligible probability, using F and F'. F4 starts by generating a key pair (SK, PK). Then, it runs

F as a subroutine and authenticates the requested streams normally using CSA. F will then output S' satisfying the conditions of this case. We also know that a continuation forger F' exists and therefore S' has a possible valid continuation. Consequently $MAC_{key}(S_j)$ is a valid MAC for some value of key, and for the conditions in this case, $key \neq h^{k-j}(\alpha)$. This all happens with probability greater than $\epsilon/24$. In order to obtain key, and thus obtain a collision for h, F4 must then run F' as a subroutine. F4 will authenticate additional streams asked by F' normally, using CSA, and will then produce a continuation of $S'^{(i)}$, as requested by F', also using CSA. When F' outputs a valid continuation T' of S', this must include the value of key, and F4 outputs $(key, h^{k-j}(\alpha))$ as a collision for h. This must occur with non-negligible probability (greater than $\epsilon * \epsilon'/24$). QED

Proof of Observation 1: We construct an algorithm F that simulates A and B over an insecure network, where E can perform active attacks. F controls the simulations of A and B out of band, i.e. over a secure, separate channel. Suppose that, with non-negligible probability, E has taken action so that B has not yet sent the first message of session SN, when A has already sent its first message of session SN. Then we construct F so that it can forge signatures under the asymmetric scheme (G,S,V). F simulates A normally, by first calling GA to generate a key pair (SK_A, PK_A). For simulating B, F does not generate a key pair, but relies on the oracle for the signature required in the first message of every session. At some point, F's simulation of A must have computed the first message of session SN, and it must therefore have received $sig_B(H, SN)$. However, we have supposed F's simulation of B has not yet computed the first message of session SN. As a consequence, $sig_B(H, SN)$ was never queried to the oracle, and can be output as a forged signature. QED.

Proof of Observation 2: Under the same setting of Observation 1, F simulates B normally, by first calling GA to generate a key pair (SK_B, PK_B). For simulating A, F does not generate a key pair, but relies on the oracle for the signature required in the first message of every session. At some point F's simulation of B must have sent the second message of session SN, and therefore it must have received $sig_A(H, SN)$. However, since F's simulation of A has not yet begun running session SN, it has not yet computed the first authenticated block. As a consequence, $sig_A(H, SN)$ was never queried to the oracle, and can be output as a forged signature. QED.

We shall now prove Theorem 1 by induction on $|S'|$, based on Lemma 1. However, we first need the following, that characterizes the information available to an active attacker at any given moment:

Lemma 2. *Suppose A and B run session SN, where:*
(1) (G,S,V) is a secure signature scheme, and
(2) h is a one-way hash function
Suppose also that B has received, during session SN, the valid authenticated stream $S' = S'_1, ..., S'_{j-1}$*, and no more. Then, there is no active attacker E that can, with non-negligible probability, cause A to release more than j authenticated blocks during session SN.*

Proof. Let parties A and B run session SN of the CSA protocol with one sender and one receiver. Suppose the Lemma is false. Then there is an active attacker E that can, with non-negligible probability, cause a situation where A has released the stream $A' = A'_1, ..., A'_{j+1}$, while B has only received $j-1$ blocks $S'_1, ..., S'_{j-1}$. Since A has released $j+1$ blocks, it must have received authenticated acknowledgements $h^k(\beta), ..., h^{k-j}(\beta)$. There are two cases, and one must hold with probability at least $\epsilon/2$:

Case 1: $\beta \neq \beta^{(SN)}$, the secret value chosen by B for session SN. Then, we show that we can construct an algorithm F1 that can simulate A and B, and use a signature oracle to forge signatures under (G,S,V). F1 simulates A by running the sender side of the CSA protocol. F1 simulates B by running the receiver side of the CSA, but uses the signature oracle to produce the signatures needed in the first authenticated block of each session. When E has caused the situation covered by this case, F1's simulation of A must have received $sig(\beta, SN)$, and since $\beta \neq \beta^{(SN)}$, this can be output as a new forged signature.

Case 2: $\beta = \beta^{(SN)}$. Then, we can construct F2 that can compute $h^{-1}(x)$, for any x, with non-negligible probability. F2 will simulate A and B, running the CSA protocol over a network where E can perform active attacks. F2 simulates A by running the sender side of the CSA protocol. F2 simulates B by running the receiver side of the CSA, but first sets the following values:

- pick SN at random between 1 and SN_{max}, where SN_{max} is the maximum number of sessions that the attacker is able to cover;
- pick j at random between 1 and j_{max}, where j_{max} is the maximum number of blocks per session in E's active attacks;
- let $k = j - 1$ and $\beta^{(SN)} = h^j(x)$;

B is then able to send to A all acknowledgments required for receiving the first $j-1$ blocks, i.e., $h^k(\beta^{(SN)}) = h^{2j-1}(x), ..., h^{k-(j-1)}(\beta^{(SN)}) = x$. When E, has caused the situation covered by this case, F2's simulation of A must have received $h^{k-j}(\beta^{(SN)}) = h^{-1}(x)$. This happens with non-negligible probability $\epsilon/2j_{max}SN_{max}$. QED.

Proof of Theorem 1: We prove a stronger claim by induction on $|S'| = j$, namely: under the conditions of the Theorem, the thesis holds and, if S'_j is non-authentic, then E can cause B to receive a valid continuation of S' only with negligible probability.

Base: We show that the claim is true for $j = 1$. Before B has received from A the first message of session SN, by Lemma 2, A has released no more than the first block of session SN. Therefore, the information available to E consists of:

- the previous authenticated streams $A'^{(1)}, ..., A'^{(SN-1)}$ sent by A, and
- the first block $A_1'^{(SN)}$ of session SN.

Let $S'_1 = S_1, MAC_1, H, SN, sig(H, SN)$. Since S' is valid, by Lemma 1, there is $q \in [1, SN]$ such that $S_1'^{(q)} = S_1^{(q)}, MAC_1^{(q)}, H, SN, sig(H, SN)$. Since sequence

number SN is only used in $A'^{(SN)}$, clearly the only value for q is SN. Also by Lemma 1, if $S'_1 \neq A'^{(SN)}_1$, then there is no polynomial algorithm that can generate a valid authenticated continuation of S', given adaptively chosen streams, and any valid continuation of A'. So neither can E, even after obtaining $A'^{(SN)}_2$.

Inductive step: Suppose that the inductive claim is true for $j-1$. We prove that it is also true for j. In order to do so, suppose B has received the valid stream $S' = (S'_1, ..., S'_j)$. This means that it was possible to produce a valid continuation of $(S'_1, ..., S'_{j-1})$, and, by the inductive hypothesis, $S_1, ..., S_{j-1}$ must be authentic with probability $1-\eta$, where η is negligible. We now have to prove that, if S_j is non-authentic, then E can cause B to receive a valid continuation of S' only with negligible probability. By Lemma 2, before B's receipt of S'_j, A has released at most j blocks during session SN. Therefore, the information available to E before B's receipt of S'_j consists of:

- the previous authenticated streams $A'^{(1)}, ..., A'^{(SN-1)}$ sent by A, and
- the stream $A'^{(SN)} = A'^{(SN)}_1, ..., A'^{(SN)}_j$ sent by A during session SN.

Let $S'_1 = S_1, MAC_1, H, SN, sig(H, SN)$. Since S' is valid, by Lemma 1, there is $q \in [1, SN]$ such that $S'^{(q)}_1 = S^{(q)}_1, MAC^{(q)}_1$, H, SN, $sig(H, SN)$. Since sequence number SN is only used in $A'^{(SN)}$, clearly the only value for q is SN. Also by Lemma 1, if $S'_j \neq A'^{(SN)}_j$, then there is no polynomial algorithm that can generate a valid authenticated continuation of S', given adaptively chosen streams, and any valid continuation of $A'^{(SN)}$. So neither can E, even after obtaining $A'^{(SN)}_{j+1}$. QED.

References

1. R. Anderson, F. Bergadano, B. Crispo, J.H. Lee, C. Manifavas, R.M. Needham, "A New Family of Authentication Protocols", *Operating Systems Review*, 32(4):9-20, 1998.
2. N. Asokan, G. Tsudik and M. Waidner, "Server Supported Signatures", *Proc.1996 Esorics*, Rome, Italy, September 1996 pp. 131-143.
3. A. Ballardie, "Scalable Multicast Key Distribution", *IETF - RFC 1949*, May 1996.
4. M. Bellare, R. Canetti and H. Krawczyk, "Pseudorandom Functions Revisited: The Cascade Construction and its Concrete Security", *Proc. 37th Symposium on the Foundations of Computer Science*, IEEE, 1996.
5. F. Bergadano and B. Crispo, "Multiparty Authentication, Fast and Short", Seminar held at IBM T. J. Watson Research Center, August 1998. http://www.research.ibm.com/security/seminar.html
6. F. Bergadano, D. Cavagnino, B. Crispo, "Individual Single Source Authentication on the MBone", IEEE International Conference on Multimedia and Expo, New York, 2000.
7. D. Bleichenbacher, U. Maurer, "On the Efficiency of One-Time Digital Signatures", in *Advances in Cryptology — AsiaCrypt 96* (Springer LNCS).
8. R. Canetti and B. Pinkas, "A Taxonomy of Multicast Security Issues", *Internet Draft*, May 1998.

9. S. Cheung, "An Efficient Message Authentication Scheme for Link State Routing", *Proc. of* 13^{th} *Annual Computer Security Applications Conference*, San Diego, California, 1997.
10. M. Dyer, T. Fenner, A. Frieze, A. Thomason, "On key storage in secure networks", *Journal of Cryptology*, vol. 8, 1995, pp. 189-200.
11. S. Even, O. Goldreich, S. Micali, "On-line / off-line digital signatures", in *Advances in Cryptology — CRYPTO 89* (Springer LNCS v. 435) pp. 263–275.
12. R. Gennaro, P. Rohatgi, "How to Sign Digital Streams", in *Advances in Cryptology — CRYPTO 97*, Springer LNCS v. 1294 pp. 180–197.
13. S. Goldwasser, S. Micali, R.L. Rivest, "A Digital Signature Scheme Secure Against Adaptive Chosen Message Attacks", in *SIAM Journal of Computing* v. 17 no. 2 (April 1988) pp. 281–308.
14. O. Goldreich, S. Goldwasser, S. Micali, "How to Construct Random Functions", *Journal of the ACM*, vol. 33:4, 1986, pp. 210-217.
15. N. M. Haller, "The S/key(tm) One-time Password System", *Proc. 1994 ISOC Symposium on Network and Distributed Security*, San Diego, CA, February 1997.
16. H. Krawczyk, M. Bellare and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", *IETF - RFC 2104*, February 1997.
17. L. Lamport, "Password Authentication with Insecure Communication", *Communication of the ACM*, 24:11, Nov. 1981, pp. 770-772.
18. R.C. Merkle, "A Digital Signature Based on a Conventional Encryption Function", in *Advances in Cryptology — CRYPTO 87* (Springer LNCS v. 293) pp. 369–378.
19. C. Metz, "Reliable Multicast: When Many Must Absolutely Positively Receive It", *IEEE Internet Computing*, pp. 9-13, July 1998.
20. S. Micali, "Enhanced Certificate Revocation System", *Technical Report MIT/LCS/TM-542* (November, 1985).
21. S. Mittra, "Iolus: a Framework for Scalable Secure Multicast", in *ACM SIGCOMM*, Cannes, September 1997.
22. A. Perrig, R. Canetti, J. D. Tygar and D. Song, "Efficient Authentication and Signing of Multicast Streams over Lossy Channels", IEEE Symposium on Security and Privacy, Oakland, California, USA, 2000.
23. R. L. Rivest and A. Shamir, "PayWord and MicroMint: Two Simple Micropayment Schemes", *Proc. 1996 Security Protocols Workshop*, Cambridge, UK, April 1996, pp. 69-87.
24. S. Schecter, T. Parnell, A. Hartemink, "Anonymous authentication of membership in dynamic groups", Proc. Workshop on Financial Cryptography, 1999.