Proceedings of the 2000 IEEE Symposium on Security and Privacy, May 2000, pp. 56-73

Efficient Authentication and Signing of Multicast Streams over Lossy Channels*

Adrian Perrig† Ran Canetti‡ J. D. Tygar† Dawn Song†

†UC Berkeley, ‡IBM T.J. Watson

{perrig,tygar,dawnsong@cs.berkeley.edu, canetti@watson.ibm.com}

Abstract

Multicast stream authentication and signing is an important and challenging problem. Applications include the continuous authentication of radio and TV Internet broadcasts, and authenticated data distribution by satellite. The main challenges are fourfold. First, authenticity must be guaranteed even when only the sender of the data is trusted. Second, the scheme needs to scale to potentially millions of receivers. Third, streamed media distribution can have high packet loss. Finally, the system needs to be efficient to support fast packet rates.

We propose two efficient schemes, TESLA and EMSS, for secure lossy multicast streams. TESLA, short for Timed Efficient Stream Loss-tolerant Authentication, offers sender authentication, strong loss robustness, high scalability, and minimal overhead, at the cost of loose initial time synchronization and slightly delayed authentication. EMSS, short for Efficient Multi-chained Stream Signature, provides non-repudiation of origin, high loss resistance, and low overhead, at the cost of slightly delayed verification.

1 Introduction

As the online population continues to expand, the Internet is increasingly used to distribute streamed media, such as streamed radio and video. We expect this trend to continue.

To enable a widespread and trusted streamed media dissemination, one must first provide sufficient security guarantees. A most prominent security risk from a user point of view is data authenticity. The user needs assurance that the data stream originated from the purported sender. Otherwise, a malicious ISP could replace parts of the stream with its own material. For example, an adversary might alter stock quotes that are distributed through IP multicast. In that scenario, the receiver needs strong sender and data authentication.

The problem of continuous stream authentication is solved for the case of one sender and one receiver via standard mechanisms, e.g. [12, 18]. The sender and receiver agree on a secret key which is used in conjunction with a message authenticating code (MAC) to ensure authenticity of each packet. In case of multiple receivers, however, the problem becomes much harder to solve, because a symmetric approach would allow anyone holding a key (that is, any receiver) to forge packets. Alternatively, the sender can use digital signatures to sign every packet with its private key. This solution provides adequate authentication, but digital signatures are prohibitively inefficient.

Real-time data streams are lossy, which makes the security problem even harder. With many receivers, we typically have a high variance among the bandwidth of the receivers, with high packet loss for the receivers with relatively low bandwidth. Nevertheless, we want to assure data authenticity even in the presence of this high packet loss.

A number of schemes for solving this problem (i.e. authenticating the data and sender in a setting where only the sender is trusted) have been suggested in the past few years [7, 13, 28, 31], but none of these schemes is completely sat-

[*] This work began in Summer 1999 when Adrian Perrig and Dawn Song were visiting the IBM T. J. Watson research lab. Initial research on stream authentication was done during Summer 1999 by Ran Canetti, Adrian Perrig, and Dawn Song at IBM. Additional improvements were suggested by J. D. Tygar in Fall 1999 at UC Berkeley. Implementation was done in Fall 1999 by Adrian Perrig at UC Berkeley. The work on stream signatures was done by J. D. Tygar, Adrian Perrig, and Dawn Song at UC Berkeley. Additional work was performed by Ran Canetti in Spring 2000. Ran Canetti is at IBM T. J. Watson Research Center, and Adrian Perrig, Dawn Song, and J. D. Tygar are at the Computer Science Division, UC Berkeley. This research was suported in part by the Defense Advanced Research Projects Agency under DARPA contract N6601-99-28913 (under supervision of the Space and Naval Warfare Systems Center San Diego), by the National Science foundation under grant FD99-79852, and by the United States Postal Service under grant USPS 1025 90-98-C-3513. Views and conclusions contained in this document are those of the authors and do not necessarily represent the official opinion or policies, either expressed or implied of the US government or any of its agencies, DARPA, NSF, USPS, or IBM.

isfactory. We discuss these schemes in section 4.

This paper presents two very different solutions to the problem of authenticating data streams efficiently in a lossy environment. The first solution, called TESLA (for Timed Efficient Stream Loss-tolerant Authentication), uses only symmetric cryptographic primitives such as pseudorandom functions (PRFs) and message authentication codes (MACs), and is based on timed release of keys by the sender. More specifically, the scheme is based on the following idea: The sender commits to a random key k without revealing it and transmits it to the receivers. The sender then attaches a message authenticating code to the next packet P_i and uses the key k as the MAC key. In a later packet P_{i+1}, the sender decommits to k, which allows the receivers to verify the commitment and the MAC of packet P_i. If both verifications are correct, and if it is guaranteed that packet P_{i+1} was not sent before packet P_i was received, then a receiver knows that the packet P_i is authentic. To start this scheme, the sender uses a regular signature scheme to sign the initial commitment. All subsequent packets are authenticated through chaining.

Our first scheme, TESLA, has the following properties:

- *Low computation overhead*. The authentication involves typically only one MAC function and one hash function computation per packet, for both sender and receiver.
- *Low per-packet communication overhead*. Overhead can be as low as 10 bytes per packet.
- *Arbitrary packet loss tolerated*. Every packet which is received in time can be authenticated.
- *Unidirectional data flow*. Data only flows from the sender to the receiver. No acknowledgments or other messages are necessary after connection setup. This implies that the sender's stream authentication overhead is independent on the number of receivers, so our scheme is very scalable.
- *No sender-side buffering*. Every packet is sent as soon as it is ready.
- *High guarantee of authenticity*. The system provides strong authenticity. By strong authenticity we mean that the receiver has a high assurance of authenticity, as long as our timing and cryptographic assumptions are enforced.[1]
- *Freshness of data*. Every receiver knows an upper bound on the propagation time of the packet.

The second scheme, called EMSS (for Efficient Multi-chained Stream Signature), is based on signing a small number of special packets in a data stream; each packet is linked to a signed packet via multiple hash chains. This is achieved by appending the hash of each packet (including possible appended hashes of previous packets) to a number of subsequent packets. Appropriate choice of parameters to the scheme guarantees that almost all arriving packets can be authenticated, even over highly lossy channels. The main features of this scheme are:

- It amortizes the cost of a signature operation over multiple packets, typically about one signature operation per 100 to 1000 packets.
- It tolerates high packet loss.
- It has low communication overhead, between 20 to 50 bytes per packet, depending on the requirements.
- It provides non-repudiability of the sender to the transmitted data.

2 TESLA: Timed Efficient Stream Loss-tolerant Authentication

In this section, we describe five schemes for stream authentication. Each scheme builds up on the previous one and improves it to solve its shortcomings. Finally, scheme V, which we call TESLA (short for Timed Efficient Stream Loss-tolerant Authentication), satisfies all the properties we listed in the introduction. The cryptographic primitives used in this section are reviewed in Appendix A, which also contains a sketch of a security analysis for our scheme.

We use the following notation: $\langle x, y\rangle$ denotes the concatenation of x and y, S stands for sender, and R stands for receiver. A stream $\mathcal{S}$ is divided into chunks M_i (which we also call messages), $\mathcal{S} = \langle M_1, M_2, \ldots, M_l\rangle$. Each message M_i is sent in a packet P_i, along with additional authentication information.

2.1 Threat Model and security guarantee

We design our schemes to be secure against a powerful adversary with the following capabilities:

- Full control over the network. The adversary can eavesdrop, capture, drop, resend, delay, and alter packets.
- The adversary has access to a fast network with negligible delay.
- The adversary's computational resources may be very large, but not unbounded. In particular, this means that

[1] However, the scheme does not provide non-repudiation. That is, the recipient cannot convince a third party that the stream arrived from the claimed source.

0-7695-0665-8/00 $10.00 © 2000 IEEE

the adversary can perform efficient computations, such as computing a reasonable number of pseudo-random function applications and MACs with negligible delay. Nonetheless the adversary cannot invert a pseudorandom function (or distinguish it from a random function) with non-negligible probability.

The security property we guarantee is that the receiver does not accept as authentic any message M_i unless M_i was actually sent by the sender. A scheme that provides this guarantee is called a *secure stream authentication scheme*.

Note that the above security requirements do not include protection against message duplication. Such protection can (and should) be added separately by standard mechanisms, such as nonces or serial numbers. Schemes I-III below do have protection against message duplication. Note also that we do not address denial-of-service attacks.

2.2 Initial synchronization (preliminary discussion)

All five schemes below begin with an initial synchronization protocol where each receiver compares its local time with that of the sender, and registers the difference. We remark that a *rough upper bound* on the clock difference is sufficient. In fact, all that the receiver needs is a value δ such that the sender's clock is no more than δ time-units ahead of the receiver's clock, where δ can be on the order of multiple seconds.[2] In section 2.8 we describe a simple protocol and discuss scalability issues related to the initial synchronization.

A basic assumption that underlies the security of our scheme is that the local internal clocks of the sender and recipient do not drift too much during a session.

2.3 Scheme I: The Basic Scheme

Here is a summary of scheme I: The sender issues a signed commitment to a key which is only known to itself. The sender then uses that key to compute a MAC on a packet P_i, and later discloses the key in packet P_{i+1}, which enables the receiver to verify the commitment and the MAC of packet P_i. If both verifications are successful, packet P_i is authenticated and trusted. The commitment is realized via a pseudorandom function with collision resistance. More details on the requirements on the pseudo-random functions are in appendix A. This protocol is similar to the Guy Fawkes protocol [1].

We now describe the basic scheme in more detail. The scheme is depicted in Figure 1. We assume that the receiver has an authenticated packet $P_{i-1} = \langle D_{i-1}, \text{MAC}(K'_{i-1}, D_{i-1})\rangle$ to start with (where $D_{i-1} = \langle M_{i-1}, F(K_i), K_{i-2}\rangle$). The fields have the following meanings. M_{i-1} is the message contained by the packet, $K'_i = F'(K_i)$ is the secret key used to compute the MAC of the next packet, and $F(K_i)$ commits to the key K_i without revealing it. The functions F and F' are two different pseudo-random functions. Commitment value $F(K_i)$ is important for the authentication of the subsequent packet P_i. To bootstrap this scheme, the first packet needs to be authenticated with a regular digital signature scheme, for example RSA [27].

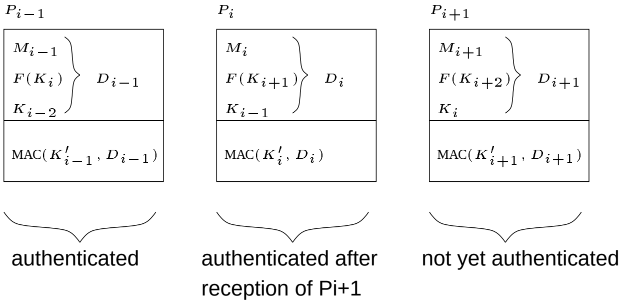

Figure 1. Basic stream authentication scheme. M_i stands for message i, P_i is packet i, K_i denotes the secret key i, F, F' are pseudo-random functions, and $\text{MAC}(K'_i, D_i)$ computes the MAC of packet i using the secret key $K'_i = F'(K_i)$.

To send the message M_i, the sender picks a fresh random key K_{i+1} and constructs the following packet $P_i = \langle D_i, \text{MAC}(K'_i, D_i)\rangle$, where $D_i = \langle M_i, F(K_{i+1}), K_{i-1}\rangle$ and the $\text{MAC}(K'_i, D_i)$ computes a message authenticating code of D_i under key K'_i.

When the receiver receives packet P_i, it cannot verify the MAC instantly, since it does not know K_i and cannot reconstruct K'_i. Packet $P_{i+1} = \langle D_{i+1}, \text{MAC}(K'_{i+1}, D_{i+1})\rangle$ (where $D_{i+1} = \langle M_{i+1}, F(K_{i+2}), K_i\rangle$) discloses K_i and allows the receiver first to verify that K_i is correct ($F(K_i)$ equals the commitment which was sent in packet P_{i-1}); and second to compute $K'_i = F'(K_i)$ and check the authenticity of packet P_i by verifying the MAC of packet P_i.

After the receiver has authenticated P_i, the commitment $F(K_{i+1})$ is also authenticated and the receiver repeats this scheme to authenticate P_{i+1} after P_{i+2} is received.

This scheme can be subverted if an attacker gets packet P_{i+1} before the receiver gets P_i, since the attacker would then know the secret key K_i which is used to compute the MAC of P_i, which allows it to change the message and the commitment in P_i and forge all subsequent traffic. To prevent this attack, the receiver checks the following *security*

[2] Many clock synchronization algorithms exist, for example the work of Mills on NTP [22], and its security analysis [5].

condition on each packet it receives, and drops the packet if the condition does not hold.

> **Security condition**: A data packet P_i arrived *safely*, if the receiver can unambiguously decide, based on its synchronized time and δ_t, that the sender did not yet send out the corresponding key disclosure packet P_j.

This stream authentication scheme is secure as long as the security condition holds. We would like to emphasize that the security of this scheme does not rely on any assumptions on network latency.

In order for the receiver to verify the security condition, the receiver needs to know the precise sending schedule of packets. The easiest way to solve this problem is by using a constant packet rate. The sending time of packet P_i is hence $T_i = T_0 + i/r$ where T_i is the time on the sender's clock and r is the packet rate (number of packets per second). In that case, the security condition which the receiver checks has the following form: $ArrT_i + \delta_t < T_{i+1}$, where $ArrT_i$ stands for the arrival time (on the synchronized receiver's clock) of packet P_i. The main problem with this scheme is that, in order to satisfy the security condition, the sending rate must be slower than the network delay from the sender to the receiver. This is a severe limitation on the throughput of the transmission. In addition, the basic scheme cannot tolerate packet loss. In particular, once a packet is dropped no further packets can be authenticated. We now gradually extend the basic scheme to eliminate these deficiencies.

2.4 Scheme II: Tolerating Packet Loss

To authenticate lossy multimedia streams, tolerating packet loss is paramount. Our solution is to generate a sequence of keys $\{K_i\}$ through a sequence generated through pseudo-random function applications. We denote v consecutive applications of the pseudo-random function F as $F^v(x) = F^{v-1}(F(x))$. By convention, $F^0(x) = x$. The sender picks a random K_n and pre-computes a sequence of n key values, where $K_0 = F^n(K_n)$, and $K_i = F^{n-i}(K_n)$. We call this sequence of values the *key chain*. Each K_i looks pseudorandom to an attacker; in particular, given K_i, the attacker cannot invert F and compute any K_j for $j > i$. On the other hand, the receiver can compute all K_j from a K_i it received, where $j < i$, since $K_j = F^{i-j}(K_i)$. Hence, if a receiver received packet P_i, any subsequently received packet will allow it to compute K_i and $K'_i = F'(K_i)$ and verify the authenticity of P_i. This scheme tolerates an arbitrary number of packet losses.

Similarly, dropping unsafe packets (i.e. those packets where the security condition does not hold) does not cause any problems in the authentication of later packets.

In the basic scheme I, an adversary might try to capture two consecutive packets before the recipient received the first of them, and then forge the packet stream. Although the security condition prevents this, the key chain also prevents this attack, because the initial commitment commits to the entire key chain and it is computationally infeasible for the attacker to invert or find collisions in the pseudo-random function.[3]

An additional benefit is that the key commitment does not need to be embedded in each packet any more. Due to the intractability of inverting the pseudo-random function, any value of the chain is a commitment for the entire chain. Hence the commitment in the initial authenticated packet is sufficient. Figure 2 shows an example of scheme II.

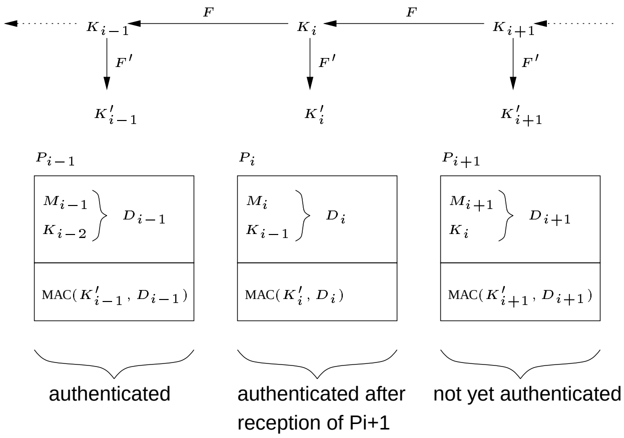

Figure 2. Scheme II. The packet format is the same as in scheme I, except that the commitment $F(K_{i-1})$ is omitted and the keys form a one-way key chain.

2.5 Scheme III: Achieving Fast Transfer Rates

As we mentioned earlier, the receiver needs to be assured that it receives the packet P_i before the corresponding key disclosure packet P_{i+1} is sent by the sender. This condition severely limits the transmission rate of the previous two schemes since P_{i+1} can only be sent after every receiver has received P_i.

We solve this problem by disclosing the key K_i of the data packet P_i in a later packet P_{i+d}, instead of in the following packet, where d is a delay parameter that is set by the sender and announced as the session set-up.

The sender determines the delay d based on the packet rate r, the maximum tolerable synchronization uncertainty

[3] I.e., it is infeasible, given $K_i = F(K_{i+1})$ to find K'_{i+1} such that $F(K'_{i+1}) = K_i$. Even if the attacker could find such a collision $F(K'_{i+1}) = K_i$ then it would be able to forge only a single message M_{i+1}. Forging additional messages would require inverting F, i.e., finding K'_{i+2} such that $F(K'_{i+2}) = K'_{i+1}$.

δ_{tMax}, and the maximum tolerable network delay d_{NMax}. Setting $d = \lceil(\delta_{\text{tMax}} + d_{\text{NMax}})r\rceil$ allows the receiver to successfully verify the security condition even in the case of maximum allowable network delay and maximal synchronization error. The choice of δ_{tMax} and d_{NMax} presents the following tradeoff: Large delay values will cause a large d which results in long delays until the packet authentication. On the other hand, short maximum delays cause the the security condition to drop packets at receivers with a slow network connection. However, multimedia data packets become obsolete if they are received after their segment of the stream was already played or presented to the user. In that case, dropping unsafe packets might not interfere with the multimedia stream since the packets are likely to be obsolete. We stress that the choice of d does not affect the security of the scheme, only its usability.

For the case of a constant packet rate, the security condition is easy to state. We assume that the sending time of the first packet is T_0 and the sending time of packet P_i is $T_i = T_0 + i/r$. To verify the security condition for an incoming packet, the receiver checks that $ArrT_i + \delta_t < T_{i+d}$, where $ArrT_i$ is the arrival time of packet P_i at the receiver.

2.6 Scheme IV: Dealing with Dynamic Packet Rates

Our previous schemes used a fixed or predictable sender schedule, with each recipient knowing the exact sending time of each packet. Since this severely restricts the flexibility of senders, we design a scheme which allows senders to send at dynamic transmission rates, without the requirement that every receiver needs to know about the exact sending schedule of each packet. The solution to this problem is to pick the MAC key and the disclosed key in each packet only on a time interval basis instead of on a packet index basis. The sender uses the same key K_i to compute the MAC for all packets which are sent in the same interval i. All packets sent in interval i disclose the key K_{i-d}.

At session set-up the sender announces values T_0 and T_Δ, where the former is the starting time of the first interval and the latter is the duration of each interval. In addition the delay parameter d is announced. These announcements are signed by the sender. The interval index at any time period t is determined as $i = \lfloor \frac{t-T_0}{T_\Delta} \rfloor$. A key K_i is associated with each interval i. The keys are chained in the same way as in Scheme II. The sender uses the same key $K'_i = F'(K_i)$ to compute the MAC for each packet which is sent in interval i. Every packet also carries the interval index i and discloses the key of a previous interval K_{i-d}. We refer to d as *disclosure lag*. The format of packet P_j is $P_j = \langle M_j, i, K_{i-d}, \text{MAC}(K'_i, M_j)\rangle$. Figure 3 shows an example of this scheme, where $d = 4$.

In this scheme, the receiver verifies the security condition as follows. Each receiver knows the values of T_0, T_Δ, and δ_t. (δ_t is the value obtained from the initial synchronization protocol.) Assume that the receiver gets packet P_j at its local time t_j, and the packet was apparently sent in interval i. The sender can be at most in interval $i' = \lfloor \frac{t_j + \delta_t - T_0}{T_\Delta} \rfloor$. The security condition in this case is simply $i + d > i'$, which ensures that no packet which discloses the value of the key could have been sent yet. Figure 4 illustrates the verification of the security condition.

It remains to describe how the values T_Δ and d are picked. (We stress that the choice of these values does not affect the security of the scheme, only its usability.) Before the sender can pick values for T_Δ and d, it needs to determine the maximum tolerable synchronization uncertainty δ_{tMax}, and the maximum tolerable network delay d_{NMax}. The sender defines $\Delta_{\text{Max}} \overset{\text{def}}{=} \delta_{\text{tMax}} + d_{\text{NMax}}$

The sender's choice for T_Δ and Δ_{Max} both present a tradeoff. First, a large value for Δ_{Max} will allow slow receivers to verify the security condition correctly, but requires a long delay for packet authentication. Conversely, a short Δ_{Max} will cause slow receivers to drop packets because the security condition is not satisfied. The second tradeoff is that a long interval duration T_Δ saves on the computation and storage overhead of the key chain, but a short T_Δ more closely achieves the desired Δ_{Max}.

After determining δ_{tMax}, d_{NMax}, and T_Δ, the disclosure lag is $d = \lceil \frac{\delta_{\text{tMax}} + d_{\text{NMax}}}{T_\Delta} \rceil$.

This scheme provides numerous advantages. First, the sender can predict how long a pre-computed key chain lasts, since the number of necessary keys is only time dependent and not on the number of packets sent. Second, the receiver can conveniently verify the security condition and the sender does not need to send its packets at specific intervals (we will discuss the details of this in Section 2.9). Another advantage is that new receivers can easily join the group at any moment. A new group member only needs to synchronize its time with the sender and receive the interval parameters and a commitment to the key chain.

2.7 Scheme V: Accommodate a Broad Spectrum of Receivers

For the previous schemes, we showed that there was a tradeoff in the choice of the key disclosure period. If the time difference is short, the packet can be authenticated quickly, but if the packet travel time is long the security condition will not hold for remote receivers, which forces them to drop the packet. Conversely, a long time period will suit remote receivers, but the authentication time delay may be unacceptable for receivers with fast network access. Since the scheme needs to scale to a large number of receivers and we expect the receivers to have a wide variety of network access, we need to solve this tradeoff. Our approach is to use multiple authentication chains (where each chain is as in scheme IV) with different disclosure periods

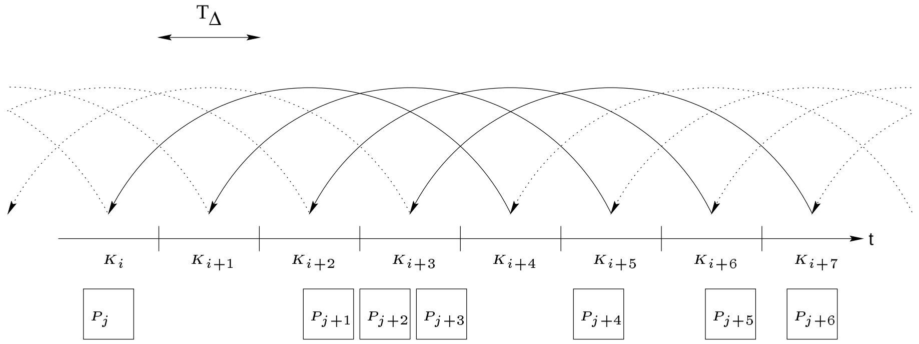

Figure 3. Scheme IV. The MAC key and disclosed key are only dependent on the time interval. The authentication key of P_j is K_i which is disclosed by packets sent during interval $i + 4$. In this case, packet P_{j+4} discloses key K_{i+1} which allows the receiver to compute K_i and to authenticate packet P_j. We would like to point out that packets P_{j+2} and P_{j+3} are both authenticated with the same MAC key K'_{i+3}, because they were sent in the same time interval.

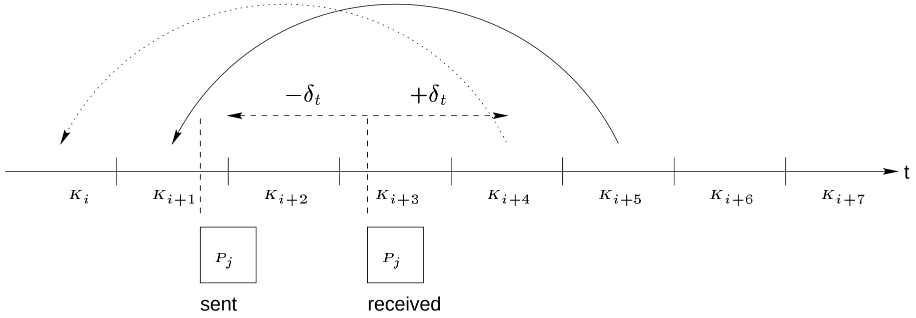

Figure 4. The security condition visualized. The packet P_j is sent in the interval where key K_{i+1} is active. The receiver receives the packet when the sender is in interval $i + 3$, but due to the δ_t the sender might already be in interval $i + 4$, which discloses key K_i. This is not a problem for the current packet, so key K_{i+1} was not disclosed yet, hence the security condition is satisfied and the packet is safe.

simultaneously. Each receiver can then use the chain with the minimal disclosure delay, sufficient to prevent spurious drops which are caused if the security condition does not hold.

The receiver verifies one security condition for each authentication chain $\mathcal{C}_i$, and drops the packet if none of the conditions are satisfied. Assume that the sender uses n authentication chains, where the first chain has the smallest delay until the disclosure packet is sent, and the nth chain has the longest delay. Furthermore, assume that for the incoming packet P_j, the security conditions for chains $\mathcal{C}_v$ $(v < m)$ are not satisfied, and the condition for chain $\mathcal{C}_m$ is satisfied. In this case, as long as the key disclosure packets for the chains $\mathcal{C}_v$ $(v < m)$ arrive, the receiver's confidence in the authenticity of packet P_j is increasing. As soon as the key disclosure packet for a chain $\mathcal{C}_v$ $(v \geq m)$ arrives, the receiver is assured of the authenticity of the packet P_j.

2.8 Initial Synchronization – Further Discussion

Our stream authentication scheme relies on a loose time synchronization between the sender and all the recipients. We call this synchronization loose, because the synchronization error can be large. The only requirement we have is that the client knows an upper bound δ_t on the maximum synchronization error.

Any time synchronization protocol can be used for our scheme, as long as it is robust against an active adversary.

As a proof-of-concept, we present a simple time synchronization protocol which suffices the requirements. The basic protocol is as follows:

$$\begin{aligned} R \to S &: \textsf{Nonce} \\ S \to R &: \{\textsf{Sender time } t_S, \textsf{Nonce}, \textsf{ Interval Rate}, \\ &\quad \textsf{Interval Id}, \textsf{ Interval start time}, \\ &\quad \textsf{Interval key}, \textsf{Disclosure Lag}\}_{K_S^{-1}} \end{aligned}$$

The receiver[4] uses a nonce in its first packet to prevent an attack which replays a previously signed synchronization reply. Besides the current time t_S at the sender, the sender also sends all information necessary to define the intervals and a commitment to the active key chain. The disclosure lag defines the difference in intervals on when the key values are disclosed. Finally, the packet is signed with a regular signature scheme.

For the purposes of our stream authentication scheme, the receiver is only interested in the maximum possible time value at the sender. This simplifies the computation. Figure 5 shows a timing diagram of the synchronization. The receiver sets $\Delta_t = t_S - t_R$ and computes the latest possible sender's time t'_S as follows: $t'_S = t'_R + \Delta_t$, where t'_R is the current receiver's time, and t'_S is the estimated sender time. In the ideal case, the receiver's initial packet arrives at the sender without delay, denoted as time t_1 in the figure. The maximum time discrepancy $\delta_t = \mathrm{RTT}$ (round-trip time).

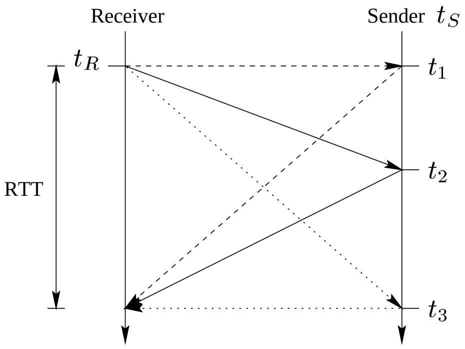

Figure 5. The receiver synchronizes its time with the sender.

Scalability is a major concern for a widely deployed system. If every receiver needs to synchronize its time with the sender, the sender could be a bottleneck. A better solution would use distributed and secure time servers. Initially, the sender synchronizes its time with the time server and computes the maximum synchronization error $\delta_t(S)$. The sender would periodically broadcast the interval information, along with its $\delta_t(S)$ and the current timestamp, digitally signed to ensure authenticity. The receivers can independently synchronize their time to the synchronization server, and individually compute their maximum synchronization error δ_t. Finally, the receivers add up all the δ_t values to verify the security condition. Taking this scheme one step further, we could have a hierarchy of synchronization servers (only the maximum errors need to propagate). We could also imagine synchronizing all the synchronization servers with a satellite signal, for example GPS.

Combining with multicast group control centers. The general IP multicast model assumes that any host can join the multicast group, receive all group data, and send data to the group [11]. To join the multicast group, the receiver only needs to announce its interest to a local router which takes care of forwarding packets to that receiver. Each joining group member contacts a central server or a group controller to negotiate access rights and session keys. This model is supported by the Secure Multicast Users Group (SMUG) [29] and we adopt it for our secure authentication scheme, which requires that each receiver performs an initial registration (for time synchronization and interval timing information) at the sender or at a central server.

Here is a sketch of a scalable synchronization mechanism that uses this infrastructure: Both senders and receivers synchronize with time synchronization servers which are dispersed in the network. After the synchronization, every entity E knows the time and the maximum error $\delta_t(E)$. The sender S periodically broadcasts a signed message which contains $\delta_t(S)$, along with the interval and key chain commitment information for each authentication chain. A new receiver R therefore only need wait for the broadcast packet allowing it to compute the synchronization error between itself and the sender as $\delta_t = \delta_t(S) + \delta_t(R)$. Based on the δ_t the receiver determines the minimum-delay authentication chain it can use. Hence, the receiver does not need to send any messages to the sender, provided that the sender and receiver have a method to synchronize and the receiver knows the upper bound of the synchronization error δ_t.

Dealing with clock drift. Our authentication protocols assume that there is no clock drift between the sender and the receiver. In practice, however, the software clock can drift (e.g. under heavy load when the timer interrupt does not get serviced). Also, an attacker might be able to change the victim's time (e.g. by sending it spoofed NTP messages). A solution to these problems is that the receiver always consults its internal hardware clock, which has a small drift and which is hard for an attacker to disturb. Furthermore, the longer authentication chains in Scheme V tolerate an authentication delay on the order of tens of seconds, giving us a large security margin. It is reasonable to assume that the hardware clock does not drift tens of seconds within one session. Finally, the receiver can re-synchronize periodically, if the hardware clock appears to drift substantially.

[4] The terms sender and receiver appear reversed in the description of this time synchronization protocol, because we keep their role with respect to the stream authentication scheme. So it is the receiver that synchronizes its time with the sender's.

2.9 Implementation Issues

We implemented a TESLA prototype in Java. We use the MD5 hash function [26] in conjunction with the HMAC construction [4] for our pseudo-random function and the MAC. To limit the communication overhead, we only use the 80 most significant bits of the output, which saves space over the standard 96 bits and gives sufficient security. The initial synchronization packet is signed using an 1024 bit RSA signature [27].

In our design, all of the functionality for TESLA remains in the application layer. This design principle follows the approach of ALF, which Tennenhouse and Clark introduce [9]. In ALF, the application knows best how to handle the data, as opposed to placing services in the network or transport layer of the OSI stack. ALF is ideally suited for TESLA. Since the authentication of packets is delayed, the application knows best how to handle unauthenticated information, which might be declared invalid later. We see two main possibilities for the application to interact with a TESLA module on the receiver side. First, we could buffer all incoming packets and deliver them only after their authenticity is assured. Second, we could deliver incoming packets directly, but inform the application through an upcall as soon as a packet is authenticated or if the packet is faulty. We implemented the second alternative.

On the other hand, however, there are also arguments for implementing TESLA in the transport layer, along with other security services [18]. Both variants of interaction with the application are possible. In the first case, the network layer buffers the stream data, and forwards it as soon as the data authenticity is guaranteed.[5] In the second case, the network layer would directly forward the data to the application, but this would require another mechanism for the network layer to inform the application about the validity of the data. To prevent applications from using data that was not authentic, we can imagine a scheme where the sender encrypts the data in each packet with a separate key and releases the key in a later packet. In this case, the application would receive the encrypted data, but could only use it after it receives the decryption key.

We use UDP datagrams for all communication to simulate multicast datagrams. We would like to point out that using a reliable transport protocol such as TCP does not make sense in this setting, because TCP interferes with the timing of packet arrival and does not announce incoming packets to the application if the previous packets did not arrive. This is a problem since our TESLA module resides in the application space. Furthermore, since TESLA is particularly well suited for lossy data streams, UDP makes perfect sense, whereas TCP is used in settings which require reliable communication.

To simplify the exposition of the protocols, we consider the case of Scheme IV, which uses one authentication chain only, as an examplar.

Sender Tasks

The sender first needs to define the following parameters for TESLA:

- The number of authentication chains
- The interval rate for each authentication chain
- The disclosure delay for each authentication chain

The number of authentication chains is dependent on the heterogeneity of network delay across receivers, the delay variance, and the desired authentication delay. For example, if we use TESLA in a LAN setting with a small network delay and low delay variance, the sender can use one single authentication chain with a disclosure lag of about one RTT, which can be as low as a few milliseconds. The other extreme is a radio broadcast over the Internet with millions of receivers. Some receivers will have high-speed network access with a low delay, others use dialup modem lines, and yet others might be connected through a wireless link with considerable delay, which can be on the order of seconds. To accommodate the latter category, which might also have a large synchronization error on the order of seconds, the longest authentication chain needs to have an disclosure delay as long as 15 to 30 seconds. Such a long delay is not acceptable to the high-speed users. A second authentication chain with a small disclosure delay around 1 - 2 seconds is appropriate. To close the wide gap between the high-end and the low-end users, a third chain with a delay of 5 to 10 seconds will appeal to the modem users.

Initially, the sender picks a random key K_n and computes and stores the entire chain of keys $K_i = F(K_{i+1})$.

Receiver Tasks

The receiver initially synchronizes with the sender and determines the accuracy δ_t. The sender also sends all interval information and the disclosure lag to the receiver, which is necessary to verify the security condition. The authenticated synchronization packet also contains a disclosed key value, which is a commitment to the key value chain.

For each incoming packet, the receiver first verifies the security condition. It then checks whether the disclosed key value is correct, which can be verified by applying the HMAC-MD5 (our pseudo-random function) until it can verify equality with a previously authenticated commitment.

[5] The argumentation against this method claims that it would put too much burden on the network layer to buffer data packet. For the case of IP fragmentation, however, the network layer already buffers data and forwards it to the application only when the entire packet is complete.

Block size	16	64	256	1024
MD5	256410	169491	72463	22075
HMAC-MD5	75187	65359	39525	17605

Table 1. Performance of primitives of the Cryptix native Java library. The performance is displayed in the number of operations per second.

To minimize the computation overhead, the receiver reconstructs and stores the chain of key values. Since the MAC cannot be verified at this time, the receiver adds the triplet $\langle \mathsf{Packet\ Hash}, \mathsf{Interval}, \mathsf{MAC\ value}\rangle$ to the list of packets to be verified, sorted by interval value. Instead of storing the entire packet, the receiver computes and stores only the hash value of the packet. If the incoming disclosed MAC key was new, the receiver updates the key chain and checks whether it can verify the MAC of any packets on the packet list. In the case a MAC does not verify correctly, the library throws an exception to warn the application. Finally, the packet is delivered to the application.

A possible denial-of-service attack is an attacker sending a packet marked as being from an interval far in the future. A receiver would then spend much time to update its key chain. A simple remedy against this attack would be for the receiver to reject packets if they could not have been sent yet (along the lines of the security condition).

A drawback of this stream authentication scheme is that each receiver needs to store the key chain and packet information to verify the packet authenticity. While the key chain is small (since only a few intervals per seconds are used in practice), the amount of storage required can be large for long authentication delays and fast sender rates. In our implementation, only the 80 bit hash and the interval are stored per packet, which amounts to 12 bytes.

Performance

For each outgoing packet, the sender only needs to compute one HMAC function per packet per authentication chain, since the key chain can be pre-computed. Table 1 shows the performance of the MD5, and HMAC-MD5 functions provided by Cryptix [10] running on a 550 MHz Pentium III Linux PC. The Java code was executed by the JIT compiler which comes with the JDK 1.1.8 provided by IBM [17].

We analyze the performance of our stream authentication scheme by measuring the number of packets per second that a sender can create. Table 2 shows the packet rates for different packet sizes and different numbers of authentication chains. We suspect that an optimized C implementation might be at least twice as fast.

Packet size (bytes)	64	256	1024
One authentication chain	27677	23009	8148
Two authentication chains	19394	14566	7402
Three authentication chains	14827	13232	6561
Four authentication chains	12653	11349	5914

Table 2. Performance of our packet authentication scheme for a varying number of authentication chains. All performance numbers are in packets per second.

The communication overhead of our prototype is 24 bytes per authentication chain. Since we use 80 bit HMAC-MD5, both the disclosed key and the MAC are 10 bytes long. The remaining four bytes are used to send the interval index.

Also, the overhead of pre-computing the key chain is minimal. In our experiments we use an interval length of $1/10$th of a second. To pre-compute a key chain long enough to authenticate packets for one hour, the sender precomputation time is only $36000/74626 \approx 0.5$ seconds.

The computational overhead on the receiver side is the same as on the sender side, except that the receiver needs to recompute the key chain while the sender can pre-compute it. However, the overhead of computing the key chain is negligible, since it involves computing one HMAC functions in each time interval, and in practice only tens of intervals are used per second.

3 EMSS: Efficient Multi-chained Stream Signature

TESLA does not provide non-repudiation. Most multimedia applications do not need non-repudiation since they discard the data after it is decoded and played. Stream signature schemes are still important, however, for the following two cases. First, some applications really do need continuous non-repudiation of each data packet, but we could not find a compelling example. Second, and more importantly, in settings where time synchronization is difficult, TESLA might not work. We present EMSS (Efficient Multi-chained Stream Signature), to achieve non-repudiation which also achieves sender authentication.

The requirements for our stream signature scheme are as follows:

- Non-repudiation for each individual packet
- Continuous non-repudiation of packets
- Robust against high packet loss

- Low computation and communication overhead
- Real-time stream content
- No buffering of packets at the sender required

3.1 Our Basic Signature Scheme

To achieve non-repudiation, we rely on a conventional signature scheme, for example RSA [27] or Rohatgi's k-times signature scheme [28]. Unfortunately, the computation and communication overhead of current signature schemes is too high to sign every packet individually. To reduce the overhead, one signature needs to be amortized over multiple packets.

Our basic solution bases on the following scheme to achieve non-repudiation of a sequence of packets. Packet P_i includes a hash $H(P_{i-1})$ of the previous packet P_{i-1}. By sending a *signature packet* at the end of the stream, which contains the hash of the final packet along with a signature, we achieve non-repudiation for all packets. To achieve robustness against packet loss, each packet contains multiple hashes of previous packets, and furthermore, the final signature packet signs the hash of multiple packets. Figure 6 shows an example, where each packet contains the hash of the two previous packets, and where the signature packet contains the hash of the last two packet and the signature.

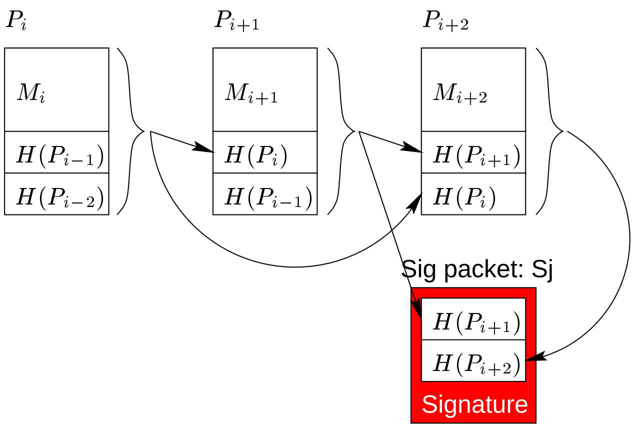

Figure 6. We achieve non-repudiation through periodic signature packets, which contain the hash of several data packets, and the inclusion of the hash of the current packet within future packets. The inclusion of multiple hashes achieves robustness against packet loss.

In order for the sender to continuously verify the signature of the stream, the sender sends periodic signature packets. Since the receiver can only verify the signature of a packet after it receives the next signature packet, it is clear that the receiver experiences a delay until packet verification.

To simplify the following discussion, we describe this scheme as a graph problem and use the corresponding terminology. Namely, we use the term node instead of packet, and edge instead of hash link. We define the length of an edge as $L(E_{ij}) = |i - j|$, where i and j are the id's of the corresponding nodes. If packet P_j contains the hash of packet P_i, we draw a directed edge starting at P_i to P_j. We call P_j a *supporting packet* of P_i. Similarly, an edge points from a packet P_k to a signature packet S_l, if S_l contains the hash of P_k. We assume that some of the packets are dropped between the sender and the receiver. All nodes which correspond to dropped packets are removed from the graph. A packet P_i is *verifiable*, if there exists a path from P_i to any signature packet S_j.

This stream signature scheme has the following parameters:

- Number of edges per node
- Length and distribution of edges
- Frequency of signature nodes
- Number and distribution of incoming edges in signature nodes

These parameters influence the computation and communication overhead, the delay until verification, and the robustness against packet loss. We want to achieve low overhead while retaining high robustness against packet loss and a low verification delay.

To simplify the problem of optimizing all parameters simultaneously, we first focus on the interplay between the number and distribution of edges to achieve high robustness against packet loss. We first consider static edges, which means that all the outgoing and incoming edges of each node have predefined lengths. For example, in a "1-3-7" scheme, the node P_i has outgoing edges to $P_{i+1}, P_{i+3}, P_{i+7}$, and incoming edges from $P_{i-1}, P_{i-3}, P_{i-7}$.

To simplify the problem even further, we initially assume independent packet loss, i.e. each packet has an equal loss probability.[6]

Instead of computing the probability precisely for each node, we wrote a program to perform simulations. We

[6] Our first attempt was to devise an analytical formula to model the probability for each node that it is connected to a signature node. Unfortunately, finding an exact formula is harder than it first appears, so deriving the analytical formula automatically for a given edge distribution remains an open problem. We illustrate this complexity with an example for the recurrence relation which describes the simple 1-2-4 scheme: $P[N-i] = (1-q)\cdot(P[N-i+1] + qP[N-i+2] + (2-q)q^2P[N-i+4] - (1-q)^2q^2P[N-i+5])$, where $P[i]$ is the probability that node i is connected to node N which is signed, and q is the probability that the node is dropped.

checked the accuracy of the simulation program on the cases for which we computed an analytical solution: $1-2-4$ and $1-2-3-4$. Our simulation (with 2500 samples simulating up to 1000 packets before the signature packet) had an absolute error of less than $\pm 2\%$ of the verification probability for these two cases.

We ran extensive simulations to find a good distribution of edges withstanding high amounts of dropped nodes. In our largest simulation, we searched through all combinations of six edges per node, where the maximum length of any edge was 51, and the probability of dropping a node was 60%.[7] In our simulation, we assumed that the final seven nodes all existed and that they all contained an edge to the signature node.

The simulation results were illuminating. The most important finding from the simulation study is that the majority of combinations are robust. Figure 8 illustrates this point. The x-axis ranges over the average probability of verification p. The figure shows how many combinations had that average verification probability p, measured over 400 nodes preceding the signature packet. The figure demonstrates that most of the combinations have high robustness. In fact, 99% of all combinations give an average verification probability over 90%. This finding motivates the use of random edges instead of static edges.

Another interesting result is that the continuous case $1-2-3-4-5-6$ is the weakest combination, and that exponentially increasing edge lengths $1-2-4-8-16-32$ had poor robustness. One of the strongest combinations is $5-11-17-24-36-39$. We show the performance of these three combinations in figure 7. The continuous case has the lowest verification probability, the exponential chain is already much better, and the last case does not seem to weaken as the distance from the signature packet increases.

The assumption of independent packet loss does not hold in the Internet. Many studies show that packet loss is correlated, which means that the probability of loss is much higher if the previous packet is lost. Paxson shows in one of his recent studies that packet loss is correlated and that the length of losses exhibit infinite variance [24]. Borella et al. draw similar conclusions, furthermore they find that the average length of loss bursts is about 7 packets [6].

Yajnik et al. show that a k-state Markov model can model Internet packet loss patterns [32]. For our simulation purposes, the two-state model is sufficient, since it can model simple patterns of bursty loss well [16, 32]. The main advantage of randomizing the edges, however, is visible when we consider correlated packet loss. Figure 9 shows a simulation with 60% packet loss and where the average length of a burst loss is 10 packets. We can clearly see in the figure that the verification probability of the static edge scheme drops exponentially, whereas the random edges still provide a high verification probability.

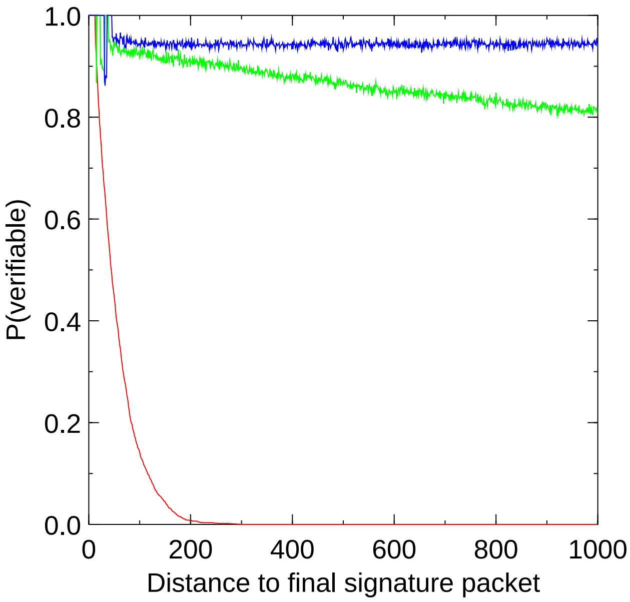

Figure 7. The verification probability for three static cases: Top line: 5-11-17-24-36-39. Middle line: 1-2-4-8-16-32. Bottom line: 1-2-3-4-5-6.

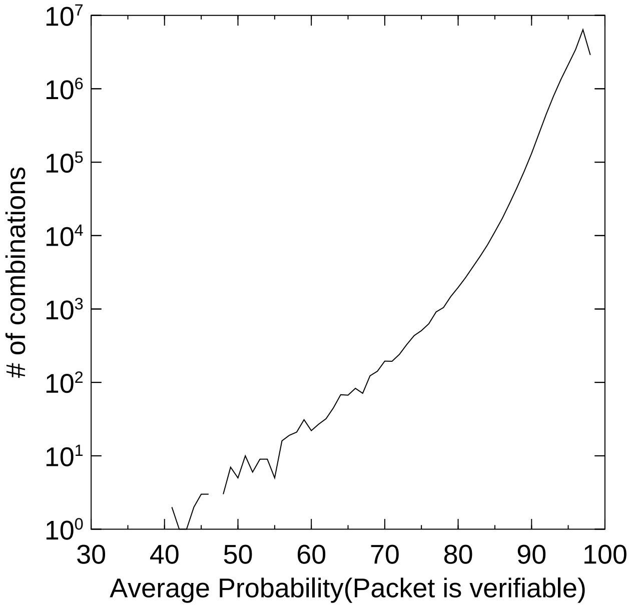

Figure 8. Number of combinations of six hashes that resulted in a given average verification probability. Note that we assume a 60% packet loss probability.

[7] We chose to use six edges per node, because we wanted to achieve a high average robustness for the case of 60% packet loss and with only five edges did not give us a high verification probability.

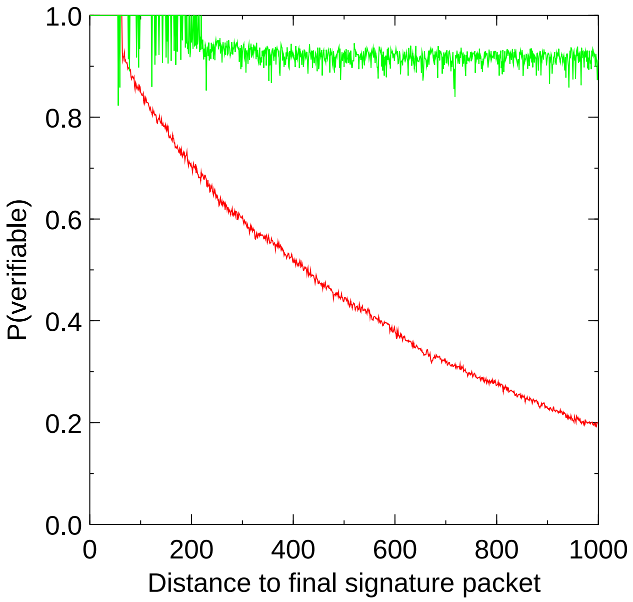

Figure 9. The verification probability for random vs a static case. Top line is random link distribution. Bottom line is 5-11-17-24-36-39.

3.2 The Extended Scheme

The basic scheme has a lot of redundancy. All the supporter packets carry the same hash value of a given packet. In the experiments we use six hashes per packet, hence six packets carry the same hash value. Removing this redundancy might give us a lower communication overhead and improved robustness against loss.

The core idea is to split the hash into k chunks, where a quorum of any k' chunks is sufficient to allow the receiver to validate the information. One approach is to use Rabin's Information Dispersal Algorithm [25], which has precisely this property. Another approach is to produce a hash function with a large number of independent bits, but only look at a limited number of those bits. This can most easily be realized by a family of universal hash functions [8].

The main advantage of this scheme is that *any* k' out of the k packets need to arrive, which has a higher robustness in some circumstances than receiving 1 packet out of d in the basic scheme. For example, if we use the basic scheme with 80-bit hashes and six hashes per packet, the communication overhead is at least 60 bytes, and the probability that at least one out of six packets arrives is $1-q^6$, where q is the loss probability. In contrast, if we use the extended scheme with a hash of 480 bits, chunks of 16 bits, $k = 30$, and $k' = 5$, the probability that the receiver gets more than four packets is $1 - \sum_{i=0}^{i<5} \binom{30}{i} \cdot q^{30-i} \cdot (1-q)^i$. Clearly, the latter probability is much higher. Although both probabilities only provide an upper bound on the verification probability, it still gives an intuition on why the extended scheme provides higher robustness to packet loss.

The simulation confirmed these findings. The extended scheme outperforms the basic scheme in robustness against packet loss. Figure 10 shows a comparison of the two schemes with identical communication overhead.

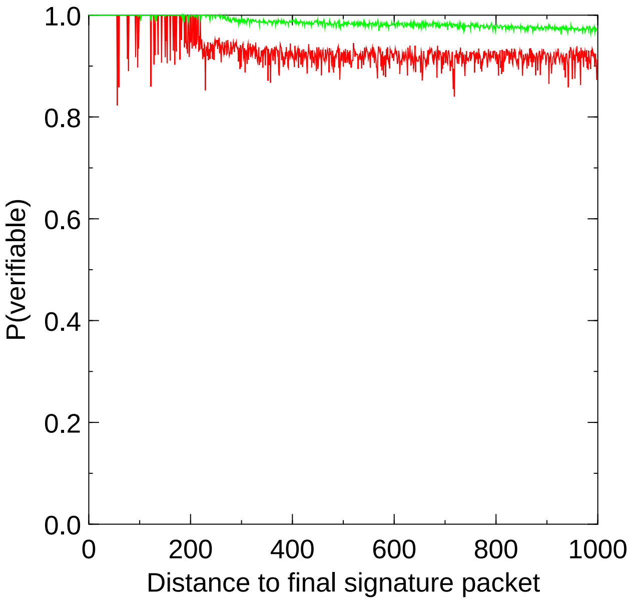

Figure 10. The verification probability for the basic vs. the extended scheme. Top line is the extended scheme. Bottom line is the basic scheme.

3.3 Signature Packets

An important requirement of our scheme signature scheme is that the receiver can continuously verify the signature of packets. Clearly, the receiver can only verify the signature once it can trace the authentication links to a signature packet. Hence, the verification delay depends on the frequency and the transmission reliability of signature packets. The signature packet rate depends on the available computation and communication resources. If we use 1024-bit RSA signatures, a dedicated server can compute on the order of 100 signatures per second. The corresponding communication overhead is 128 bytes for the signature plus 10 bytes for each hash included.

We also performed simulations with signature packets. The parameters included the signature rate, the loss probability of signature packets,[8] and the number of hashes per signature packet. Figure 11 shows the sawtooth-shaped

[8] The loss probability might be different for signature packets if they are sent redundantly or in a higher service class in the context of QoS.

verification probability for a stream with 10% packet loss (bursty loss), the average burst length of dropped packets is 10, the hash is split up into 9 chunks of 27 bits each (spanning a maximum length of 100 packets), hence 3 chunks are necessary to verify a packet, which gives us 81 bits of the signature. The communication overhead per packet is therefore about 35 bytes per packet. The signature packets are sent every 250 packets and they contain 80-bit hashes of 40 packets, and one 1024-bit RSA digital signature which amounts to 128 bytes. Each signature packet is sent twice, so the loss probability of a signature packet is reduced to 1%. The average per-packet overhead in this case is 40 bytes.

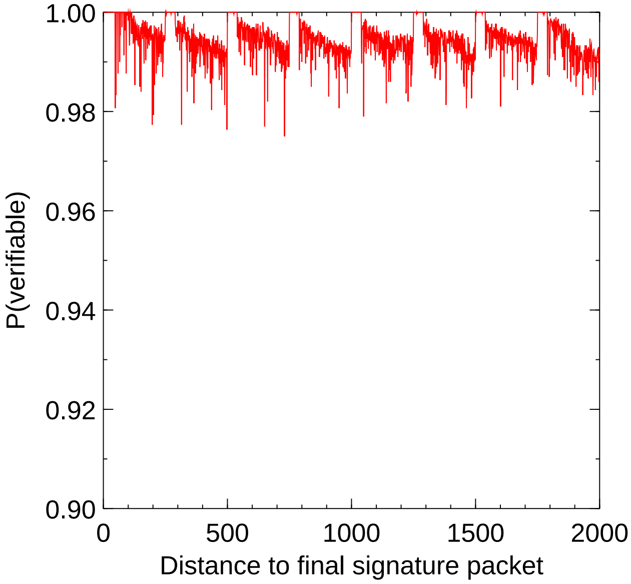

Figure 11. The verification probability for the extended scheme including periodic signature packets.

3.4 Case Study on Two Settings

We consider two different cases of stream distribution and we analyze the overhead of applying EMSS to ensure the non-repudiation of the streamed data.

Case I: Streamed Distribution of Traffic Data

Assume that a municipality has traffic sensors distributed over streets. It broadcasts this data over the Internet so citizens (and robot driven vehicles) can improve their trip planning. The system requirements are as follows:

- The data rate of the stream is about 8 Kbps, about 20 packets of 64 bytes each are sent every second.
- The packet drop rate is at most 5%, where the average length of burst drops is 5 packets.
- The verification delay should be less than 10 seconds.

Many different instantiations of EMSS result in efficient schemes which satisfy these requirements. The following scheme offers low overhead with high verification probability. Each packet has two hashes, and the length of each hash chain element is chosen uniformly distributed over the interval $[1, \ldots, 50]$. Each hash is 80 bits long, hence, only one hash is necessary for verification. A signature packet is sent every 100 packets, or every five seconds, which is not necessary to achieve robustness in this case, but to ensure that the verification delay is less than ten seconds, with high probability. Each signature packet carries the hash of five data packets. The simulation predicts an average verification probability per packet of 98.7%.

The computation overhead is minimal. The sender only needs to compute one signature every five seconds, and only 20 hash functions per second. The communication overhead is low also. Each data packet carries 20 bytes containing the hash of two previous packets.[9] The signature packet contains five hashes and a signature, and its length is hence 50 bytes plus the signature length. Assuming a 1024 bit RSA signature, the signature packet is 178 bytes long. The average per-packet overhead is therefore about 22 bytes, which is much lower than previous schemes, which we review in section 4.

Case II: Real-time Video Broadcast

Assume we want to broadcast signed video on the Internet. The system requirements are as follows:

- The data rate of the stream is about 2 Mbps, about 512 packets of 512 bytes each are sent every second.
- Some clients experience packet drop rates up to 60%, where the average length of burst drops is 10 packets.
- The verification delay should be less than 1 second.

The high packet drop rate makes it difficult for signature packets to reach the receiver. To increase the likelihood of signature packets to arrive, we send them twice — but within a delay, since packet loss is correlated. If we approximate the loss probability by assuming the signature

[9] The packet id's of the packet do not need to be stored in the packet for two reasons. Since the probability of a hash collision is negligible, the receiver can store the hash of the last 50 data packets it received. If any packet contains the same hash value, we consider that packet as verified, if the current packet can be verified. Alternatively, we could build a deterministically computable random graph over the packets, and the receiver would reconstruct it. This alternative would require a packet id in each packet.

packet losses are uncorrelated if they are sent within a delay, the probability that one of them arrives is approximately $1 - 0.6^2 = 0.64$. Since the packet loss is so high and verification delay relatively short, we send a a signature packet every 200 packets. This translates to about 2.5 signatures per second, which we consider as a low computational overhead. We assume that the signature packets have about the same size as the data packets, so in 512 bytes we can fit one 1024-bit RSA signature and the 80 bit hash of 40 previous packets.

We chose these parameters based on good engineering practice. To find better parameters for the number of chunks that the hash is split into and the number of chunks required to verify the packet, we used a simulation. The simulation shows that the best combination for this case uses 50 bytes per packet to insert 25 chunks of two bytes of the hash of previous packets. Including the signature packets, the average communication overhead is about 55 bytes per packet. The simulation predicts the average verification probability over the final 2000 packets of 97%, with the minimum verification probability 90%.

4 Previous Work

We review previous art which deals with the problem of continuous authentication and signature of streams.

Gennaro and Rohatgi introduced techniques for signing digital streams [13]. They present two different schemes, one for the off-line case (the entire stream content is known in advance) and the other for the on-line case (the stream content is generated in real-time). For the off-line case, they suggest signing the first packet and embeding in each packet P_i the hash of the next packet P_{i+1} (including the hash stored in P_{i+1}). While this method is elegant and provides for a stream signature, it does not tolerate packet loss. The biggest disadvantage, however, is that the entire stream of packets needs to be known in advance. The on-line scheme solves this problem through a regular signature of the initial packet and embedding the public key of a one-time signature in each packet, which is used to sign the subsequent packet. The limitation is again that this scheme is not robust against packet loss. In addition, the one-time signature communication overhead is substantial.

Wong and Lam address the problem of data authenticity and integrity for delay-sensitive and lossy multicast flows [31]. They propose to use Merkle's signature trees to sign streams. Their idea to make asymmetric digital signatures more efficient is to amortize one signature generation and verification over multiple messages. Merkle describes how to construct a hash tree over all messages where the signer only digitally signs the root [20, 21]. However, to make this scheme robust against packet loss, every packet needs to contain the signature along with all the nodes necessary to compute the root, which requires large space overhead. In practice, this scheme adds around 200 bytes to each packet (assuming a 1024 bit RSA signature and a signature tree over 16 packets). Another shortcoming is that all messages need to be known to compute the signature tree. This causes delays on the sender side. Furthermore, after the signature computation, all packets are sent at the same time, causing bursty traffic patterns. This burstiness may increase the packet drop rate in the network. Although the computational overhead is amortized over multiple packets, there is still a substantial amount of computation necessary for signature verification, which can consume a substantial amount of resources on low-end receivers (for example battery power). A subtle point is that the per-packet computation increases with the packet loss rate. Since mobile receivers also have less computational power and higher packet loss, the benefit of the amortization is lost. The schemes which we propose in this paper solve these shortcomings.

Rohatgi presents a new scheme which reduces the sender delay for a packet, and which reduces the communication overhead of one-time signatures over previously proposed schemes [28]. He introduces a k-time signature scheme, which is more space efficient than the one-time signatures. Despite all advantages, the scheme still uses 90 bytes for a 6-time public key (which does not include the certificate of the public key) and 300 bytes for each signature. Also, the server requires 350 off-line hash function applications and the client needs 184 hashes on average to verify the signature.

Canetti et al. construct a sender authentication scheme for multicast [7]. Their solution is to use k different keys to authenticate every message with k different MAC's. Every receiver knows m keys and can hence verify m MAC's. The keys are distributed in such a way that no coalition of w receivers can forge a packet for a specific receiver. The communication overhead for this scheme is considerable, since every message carries k MAC's. The server must also compute k MACs before a packet is sent, which makes it more expensive than the scheme we present in this paper. Furthermore, the security of their scheme depends on the assumption that at most a bounded number (which is on the order of k) of receivers collude.

Syverson, Stubblebine, and Goldschlag propose a system which provides asymmetric and unlinkable authentication [30]. In their system, a client proves its right to access the vendor's service through a blinded signature token, which is renewed on each transaction. Through the vendor's blind signature, they achieve unlinkability of transactions. This scheme would not work for stream authentication, because the communication and computation overhead is substantial. Furthermore, the scheme provides unlinkability, which is not needed for authenticating multicast streams.

Anderson et al. [1] present a scheme which provides stream authentication between two parties. Their Guy Fawkes protocol has the following packet format: $P_i = \{M_i, X_i, h(X_{i+1}), h(M_{i+1}, X_{i+1}, h(X_{i+2}))\}$, where M_i denotes message i, X_i stands for a random number, and h is a hash function. Assuming that the receiver received an authentication packet P_i, it can immediately authenticate the following packet P_{i+1}, since P_i contains the commitment $h(M_{i+1}, X_{i+1}, h(X_{i+2}))$ to P_{i+1}. Similarly, P_{i+1} comes with a commitment for P_{i+2}. A drawback of this protocol is that to send message M_i, the following message M_{i+1} needs to be known. Furthermore, this scheme cannot tolerate any packet loss. They propose two methods to guarantee that the keys are not revealed too soon. The first method is that the sender and receiver are in lockstep, i.e. the receiver acknowledges every packet before the sender can send the next packet. This severely limits the transfer time and does not scale to a large number of receivers. The second method to secure their scheme is to time-stamp each packet at a time-stamping service, which introduces additional complexity. The Basic authentication scheme I we propose in this paper is similar to the Guy Fawkes protocol. We improve on Guy Fawkes and construct an efficient stream authentication scheme without these limitations.

We understand that unpublished work by Bob Briscoe at BT research, and Dan Boneh and Philippe Golle, has been proceeding along some similar lines. To the best of our knowledge, all of these groups have been working independently.

5 Acknowledgments

We would like to thank Pankaj Rohatgi for his help during the early stages of the project. We would also like to thank Steve Glassman, Mark Manasse, and Allan Heydon for their helpful comments and discussions. We are also indebted to David Wagner and Bob Briscoe for their relevant feedback and concrete suggestions on how to improve the presentation of this work. Finally, we thank the anonymous reviewers for their helpful suggestions.

References

[1] Ross J. Anderson, Francesco Bergadano, Bruno Crispo, Jong-Hyeon Lee, Charalampos Manifavas, and Roger M. Needham. A new family of authentication protocols. *Operating Systems Review*, 32(4):9–20, October 1998.

[2] M. Bellare, J. Kilian, and P. Rogaway. The security of cipher block chaining. In Yvo Desmedt, editor, *Advances in Cryptology - Crypto '94*, pages 341–358, Berlin, 1994. Springer-Verlag. Lecture Notes in Computer Science Volume 839.

[3] M. Bellare and P. Rogaway. Collision-resistant hashing: Towards making UOWHFs practical. In Burt Kaliski, editor, *Advances in Cryptology - Crypto '97*, pages 470–484, Berlin, 1997. Springer-Verlag. Lecture Notes in Computer Science Volume 1294.

[4] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Message Authentication using Hash Functions — The HMAC Construction. *RSA Laboratories CryptoBytes*, 2(1), Spring 1996.

[5] Matt Bishop. A Security Analysis of the NTP Protocol Version 2. In *Sixth Annual Computer Security Applications Conference*, November 1990.

[6] M. Borella, D. Swider, S. Uludag, and G. Brewster. Internet packet loss: Measurement and implications for end-to-end qos. In *International Conference on Parallel Processing*, August 1998.

[7] Ran Canetti, Juan Garay, Gene Itkis, Daniele Micciancio, Moni Naor, and Benny Pinkas. Multicast security: A taxonomy and some efficient constructions. In *Infocom '99*, 1999.

[8] J. L. Carter and M. N. Wegman. Universal classes of hash functions. *JCSS No. 18*, (18):143–154, 1979.

[9] D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. In *Proceedings of the ACM symposium on Communications architectures and protocols SIGCOMM '90*, pages 200–208, September 26-28 1990.

[10] Cryptix. `http://www.cryptix.org`.

[11] Stephen E. Deering. Multicast Routing in Internetworks and Extended LANs. In *Proceedings of ACM SIGCOMM '88*, August 1988.

[12] T. Dierks and C. Allen. The TLS protocol version 1.0. Internet Request for Comments RFC 2246, January 1999. Proposed standard.

[13] Rosario Gennaro and Pankaj Rohatgi. How to Sign Digital Streams. Technical report, IBM T.J.Watson Research Center, 1997.

[14] Oded Goldreich. Foundations of cryptography (fragments of a book). `http://www.toc.lcs.mit.edu/\~oded/frag.html`, 1998.

[15] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal of Computing*, 17(2):281–308, April 1988.

[16] Mark Handley. Private communication with Adrian Perrig, February 2000.

[17] IBM. Java web page. http://www.ibm.com/developer/java.

[18] Ipsec. IP Security Protocol, IETF working group. http://www.ietf.org/html.charters/ipsec-charter.html.

[19] Michael George Luby. *Pseudorandomness and Cryptographic Applications*. Princeton Computer Science Notes, 1996.

[20] R. C. Merkle. A certified digital signature. In Gilles Brassard, editor, *Advances in Cryptology - Crypto '89*, pages 218–238, Berlin, 1989. Springer-Verlag. Lecture Notes in Computer Science Volume 435.

[21] Ralph Merkle. Protocols for public key cryptosystems. In *1980 IEEE Symposium on Security and Privacy*, 1980.

[22] David L. Mills. Network Time Protocol (Version 3) Specification, Implementation and Analysis. Internet Request for Comments, March 1992. RFC 1305.

[23] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing (STOC '89)*, 1989.

[24] V. Paxson. End-to-end internet packet dynamics. *IEEE/ACM Transactions on Networking*, 7(3):277–292, June 1999.

[25] M. O. Rabin. The information dispersal algorithm and its applications, 1990.

[26] Ronald L. Rivest. The MD5 message-digest algorithm. Internet Request for Comments, April 1992. RFC 1321.

[27] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[28] Pankaj Rohatgi. A compact and fast hybrid signature scheme for multicast packet authentication. In *6th ACM Conference on Computer and Communications Security*, November 1999.

[29] Secure Multicast User Group (SMUG). http://www.ipmulticast.com/community/smug/.

[30] Paul F. Syverson, Stuart G. Stubblebine, and David M. Goldschlag. Unlinkable serial transactions. In *Financial Cryptography '97, Springer Verlag, LNCS 1318*, 1997.

[31] C. K. Wong and S. S. Lam. Digital signatures for flows and multicasts. In *Proc. IEEE ICNP '98*, 1998.

[32] M. Yajnik, S. Moon, J. Kurose, and D. Towsley. Measurement and modelling of the temporal dependence in packet loss. In *IEEE INFOCOM '99*, New York, NY, March 1999.

A Proof of Security

In this appendix, we present a more formal statement of the security assumptions on our cryptographic primitives and sketch the proof of security for one of our stream authentication schemes. First, here are primitives we use in our schemes.

Message Authentication Codes (MACs). A function family $\{f_k\}_{k\in\{0,1\}^\ell}$ (where ℓ is the key length, taken to be the security parameter) is a secure MAC family if any adversary A (whose resources are bounded by a polynomial in ℓ) succeeds in the following game only with negligible probability. A random ℓ-bit key k is chosen; next A can adaptively choose messages $m_1, \ldots, m_n$ and receive the corresponding MAC values $f_k(m_1)\ldots f_k(m_n)$. A succeeds if it manages to forge the MAC, i.e., if it outputs a pair m, t where $m \neq m_1, \ldots, m_n$ and $t = f_k(m)$. See [2] for more details.

Pseudorandom functions (PRFs). A function family $\{f_k\}_{k\in\{0,1\}^\ell}$ (where ℓ is the key length, taken to be the security parameter) is a pseudorandom function family if any adversary A (whose resources are bounded by a polynomial in ℓ) cannot distinguish between a function f_k (where k is chosen randomly and kept secret) and a totally random function only with negligible probability. That is, a function g is chosen to be either f_k for a random ℓ-bit key, or a random function with the same range. Next A gets to ask the value of g on as many points as it likes. Nonetheless A should be unable to tell whether g is random or pseudorandom. see [14, 19] for more details.

The schemes below make use of the following property of pseudorandom functions: as long as the key k is random (or pseudorandom) and remains unknown, the value $k_1 = f_k(x)$ is also pseudorandom for any fixed and known x. (In our schemes we use the arbitrary value $x = 0$.) This allows us to securely iterate; that is, $k_2 = f_{k_1}(x)$ is also pseudorandom, and so on. Furthermore, the value

0-7695-0665-8/00 $10.00 © 2000 IEEE

$k'_1 = f_k(x')$ where $x \neq x'$ is cryptographically independent from k_1 (as long as k remains secret) and can be used as a key for different cryptographic transforms (such as a MAC).

Target collision resistance. A function family $\{f_k\}_{k \in \{0,1\}^\ell}$ (where ℓ is the key length, taken to be the security parameter) is Target Collision Resistant if any adversary A (whose resources are bounded by a polynomial in ℓ) can win in the following game only with negligible probability. First A generates a value v_1 in the common domain of $\{f_k\}$. Next an ℓ-bit key k is randomly chosen and *given to* A. Next A wins if it generates v_2 such that $f_k(v_1) = f_k(v_2)$. Note that target collision resistance implies *2nd pre-image collision resistance.* See more details in [3, 23].

In our scheme we use a PRF family $\{f_k\}$ that also has the following flavor of target collision resistance. First a key k is chosen at random, and the adversary is given $f_k(0)$. Next the adversary is assumed to be unable (except with negligible probability) to find $k' \neq k$ such that $f_{k'}(0) = f_k(0)$.

Since any PRF family is also a secure MAC family, in our schemes we use the same function family for both purposes. Still, for clarity, in the sequel we differentiate between the cryptographic functionality of a PRF and a MAC.[10]

In addition, we use digital signatures (secure against chosen message attacks, see [15]), where the sender holds the signing key and all receivers hold the corresponding public verification key. The way in which the receivers obtain the verification key is left out of scope.

Security Analysis of Scheme III

For brevity, we only sketch a proof of security of one of the TESLA schemes, specifically Scheme III.

Theorem A.1. *Assume that the PRF, the MAC and the signature schemes in use are secure, and that the PRF has the TCR property described in Section A. Then Scheme IV is a secure stream authentication scheme.*

Proof sketch. For simplicity we assume that the MAC and the PRF are realized by the same function family $\{f_k\}$. (In our implementation, f = HMAC.) Assume for contradiction that Scheme III is not a secure stream authentication scheme. This means that there is an adversary A who controls the communication links and manages, with non-negligible probability, to deliver a message m to a receiver R, such that the sender S has not sent m but R accepts m as authentic and coming from S.

We show how to use A to break the security of one of the underlying cryptographic primitives in use. Specifically, we construct a distinguisher D that uses A to break the security of the function family $\{f_k\}$. That is, D gets access to a black-box g and can tell with non-negligible probability if g is a function f_k where k is a random and secret key, or if alternatively g is a totally random function. For this purpose, D can query g on inputs x of its choice and be answered with $g(x)$.

Distinguisher D works by running A, as follows. Essentially, D simulates for A a network with a sender S and a receiver R. That is:

1. D chooses a number $\ell \in \{1..M\}$ at random, where M is the total number of messages to be sent in the stream. (D hopes that A will forge the ℓth message, m_ℓ.)

2. D chooses signing and verification keys for S, and hands the verification key to A.

3. D hands to A the initial message from S. This message is signed using S's signing key, and contains the key K_0, plus the starting time T_0 and the duration d of a time interval. The key K_0 is generated as in the scheme, with the following twist: Recall that in the the scheme $K_0 = F^n(K_n)$ where $F^i(x) = f_{F^{i-1}(x)}(0)$ and K_n is a randomly chosen value (with the appropriate length). Here, $K_0 = F^{\ell-1}(K_{\ell-1})$ where $K_{\ell-1} = g(0)$.

4. For the first $\ell - 1$ messages in the stream D runs the sender's algorithm in Scheme III with no modifications. Whenever a message m_i (with $i < \ell$) is generated, it is handed to A.

5. Message m_ℓ is generated as in Scheme III, with the following exception: In the scheme, the MAC in m_ℓ should equal $f_{K_\ell}(M_\ell, K_{\ell-1})$ where M_ℓ is the actual data in message m_ℓ. Here, D lets the MAC be $g(M_\ell, K_{\ell-1})$.

6. D inspects the messages that A delivers to the receiver R from the moment A receives m_ℓ and until time $T_0 + \ell \cdot d$. (All times are taken locally within D.) If A delivers a message m' that is different than m_ℓ and has a valid MAC with respect to g (i.e., m' is of the form $m' = (M', K', g(M', K'))$) D decides that g was chosen from the pseudorandom family $\{f_k\}$. Otherwise (i.e., A does not successfully forge a message) D decides that g is a random function.

We sketch the argument demonstrating that D succeeds with non-negligible probability. If g is a truly random function then A has only negligible probability to successfully

[10] In fact, we do not need the full security guarantee of a PRF. It suffices to have a (length-doubling) pseudorandom generator with a similar TCR property to the one described above. Nonetheless, for simplicity we describe our schemes as ones using a full-fledged PRF.

forge the ℓth message in the stream. Therefore, if g is random then D makes the wrong decision only with negligible probability.

On the other hand, we have assumed that if the authentication is done using $\{f_k\}$ then A forges some message with non-negligible probability ϵ. It follows that A forges the ℓth message with probability at least ϵ/ℓ. Furthermore, our timing assumption guarantees that A does so prior to time $T_0 + \ell \cdot d$. It follows that if g is taken from $\{f_k\}$ then D makes the right decision with probability at least ϵ/ℓ (which is non-negligible).

We remark that the above argument fails if A hands R a forged initial message from S, or if for some $i < \ell$ adversary A finds a key K_i' that is different than K_i, before time $T_0 + i \cdot d$. However, in these cases the security of the signature scheme or the target collision resistance of $\{f_k\}$ is compromised, respectively. □