Internet Engineering Task Force IRTF SMUG
Internet Draft Perrig, Canetti, Briscoe, Tygar, Song
draft-irtf-smug-tesla-00.txt UC Berkeley / Digital Fountain / IBM / BT
17 November 2000
Expires: 17 June 2001

TESLA: Multicast Source Authentication Transform

STATUS OF THIS MEMO

Abstract

This document describes TESLA, a secure sender authentication mechanism for multicast or broadcast data streams. Data authentication is an important component for many applications, for example audio and video Internet broadcasts, or data distribution by satellite.

The main deterrents so far for a data authentication mechanism for multicast were the seemingly conflicting requirements: loss tolerance, high efficiency, no per-receiver state at the sender. The problem is particularly hard in settings with high packet loss rates and where lost packets are not retransmitted, and where the receiver wants to authenticate each packet it receives.

TESLA provides authentication of individual data packets, regardless of the packet loss rate. In addition, TESLA features low overhead for both sender and receiver, and does not require per-receiver state at the sender. TESLA is secure as long as the sender and receiver are loosely time synchronized.

Table of Contents

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, an attacker 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. [1,2]. 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 high packet loss. One of the main challenges is to ensure data authenticity of every received packet, even though lost packets are not retransmitted.

1.1 Previous Work

A number of schemes for solving data authentication have been suggested in the past few years [3,4,5,6,7,8,9,10,11]. An Internet Draft based on [7] was proposed by McCarthy in 1998 but was not updated.

This document is based mainly on the TESLA [8,9] and FLAMeS [10] schemes, which have low computation and communication overhead. Similar schemes were suggested by Cheung [12], and by Bergadano et al. [11,13].

1.2 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [14].

2 Rationale

The authentication of broadcast data is an important building block for settings that require source authentication.

The secure multicast group (SMuG) views multicast source authentication as one of the fundamental security services [15]. In particular, an early SMuG draft mentions the multicast source authentication and data integrity building block as one of four fundamental functional building blocks [16].

The recent multicast data security transformation draft also needs source authentication [17]. The draft distinguishes between two data authentication methods, namely the internal data authentication tag, and the external authentication data. The former is encrypted with the payload and the latter is outside of the encrypted data. TESLA can be used in both cases. In this document, we propose TESLA as a building block. We define the format of the authentication information only, without committing to any specific packet format.

Source authentication is also required within the reliable multicast transport (RMT) IETF group [18]. In particular, authentication is required in the Asynchronous Layered Coding (ALC) draft [19]. TESLA is ideally suited also in this setting to provide source authentication, possibly by incorporating it within AMESP and then in turn incorporating AMESP within the ALC header. See details in [17].

3 Functionality

TESLA provides delayed per-packet data authentication. The key idea to provide both efficiency and security is a delayed disclosure of keys. The delayed key disclosure results in an authentication delay. In practice, the delay is on the order of one RTT (Round-trip-time).

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 12 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 except for time synchronisation or for eventual initial connection setup. This implies that the sender's stream authentication

overhead is independent of the number of receivers, hence TESLA 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. 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.

TESLA can be used both in the network layer or in the application layer. The delayed authentication, however, requires buffering of packets until authentication is completed.

3.1 Threat Model and Security Guarantee

We design TESLA 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.

- 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 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 pseudo-random function (or distinguish it from a random function) with non-negligible probability.

The security property TESLA guarantees is that the receiver never accepts M_i as an authentic message unless M_i was actually sent by the sender. A scheme that provides this guarantee is called a secure stream authentication scheme.

3.2 Assumptions

For TESLA to be secure, the sender and the receiver ARE REQUIRED to be loosely time synchronized. Loosely time synchronized means that the synchronization does not need to be precise, but the receiver MUST know an upper bound on the dispersion (the maximum clock

offset). TESLA supports two synchronization methods, direct and indirect synchronization. In direct synchronization, the receiver synchronizes its time directly with the data sender, in indirect synchronization both the sender and the receiver synchronize their time with a common time synchronization service. If the latter method is used, we assume that secure time synchronization servers exist in the network, for example NTP servers [20]. Finally, the clock of the receiver MUST have known drift while receiving information from the server and MUST be secure against alteration by an attacker. We would like to emphasize that the security of TESLA does not rely on any assumptions on network propagation delay.

TESLA also MUST be bootstrapped through a regular data authentication system. We recommend to use a digital signature algorithm for this purpose, in which case the receiver is REQUIRED to have an authentic copy of either the sender's public key certificate or a root key certificate in case of a PKI (public-key infrastructure).

Finally, the MAC and pseudo-random functions MUST be cryptographically secure. Further details on the instantiation of the MAC and PRF are in section 6.3.

4 Notation

To denote the subscript or an index of a variable, we use the underscore between the variable name and the index, e.g. the key K with index i is K_i, the key K with index i+d is K_{i+d}. To write a superscript we use the caret, e.g. the function F with the argument x executed i times is F^i(x), executed j-1 times we write F^{j-1}(x).

5 Detailed TESLA Description

In this section, we describe TESLA in detail. We first discuss our threat model and the security guarantees that TESLA provides. Next we describe the sender setup, the receiver bootstrapping process, followed by the detailed sender receiver operations.

5.1 Sender Setup

In our model, a sender distributes a long message M as a stream of small message chunks M_i (the i'th chunk of the message). Generally, the sender sends each message chunk M_i in one network packet P_i.

For the purpose of TESLA, the sender splits the time up into even intervals I_i, where the duration of each interval is Tint and the starting time of an interval is TI_i. Trivially, we have TI_i = TI_0 + i * Tint. In each interval, the sender may send zero or multiple packets.

Before sending the first message, the sender determines the sending duration (possibly infinite), the interval duration, and the number N of keys of the key chain. In section 6.1 and 6.2, we will go into more detail about the choice of Tint and on the length of the hash chain. This key chain is analogous to the one-way chain introduced by Lamport [21], and the S/KEY authentication scheme [22]. The sender then picks the last key K_N of the key chain randomly and pre-computes the entire key chain using a pseudo-random function F, which is by definition a one-way function. Each element of the chain is defined as K_i = F(K_{i+1}). Each key can be derived from K_N as K_i = F^{N-i}(K_N), where F^j(k) = F^{j-1}(F(k)) and F^0(k) = k. Each key of the key chain corresponds to one interval (K_j is active in interval I_j).

Since we do not want to use the same key multiple times in different cryptographic operations, we use a second pseudo-random function F' to derive the key which is used to compute the MAC of messages in each interval. Hence, K'_i = F'(K_i). Figure 1 depicts this key derivation. The choice of F and F' are detailed in section 6.3.

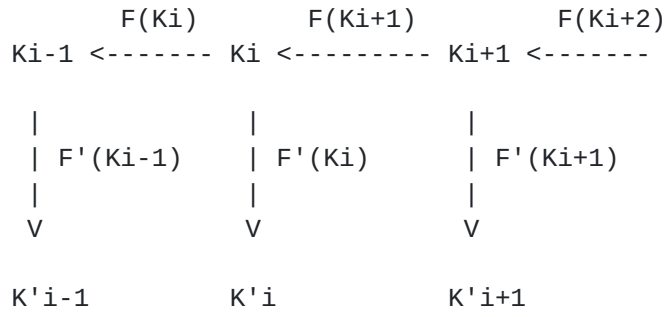

Figure 1: TESLA key chain and the derived MAC keys

5.2 Bootstrapping a new Receiver

TESLA requires loosely synchronized clocks between the sender and the receivers. Furthermore, TESLA requires an initially authenticated data packet. This authentication is achieved with a digital signature scheme, such as RSA [23], or DSA [24].

We present two options for synchronizing the time. Either the receiver directly synchronizes its time with the sender (direct synchronization), or both the sender and receiver synchronize their time with a third entity, a time server (indirect synchronization), for

example using NTP [20]. Both schemes (including their packet format) are described in appendix B. Whatever time synchronization mechanism is used, the receiver MUST know the dispersion (maximum clock offset) after the time synchronization.

For both schemes, we assume that the sender's and receiver's clocks have negligible drift, otherwise the receiver uses periodic re-synchronization, or accounts for the drift with an increased d_t.

The receiver MUST receive the following authenticated information about the time intervals and key chain:

- The beginning time of a specific interval TI_j
- The id of that interval I_j
- The interval duration Tint
- The number n of authentication instances per packet. Note that all TESLA instances share the same time interval and key chain, only the key disclosure delay is different for each instance.
- Key disclosure delay d_v of each authentication instance (unit is interval) (1 <= v <= n)
- A publically known key K_i (i < j - d where j is the current interval index)
- The interval of the last key chain element (key chain expiration)
- Total key chain length
- Indicate if immediate authentication is used, and how many hashes of future data segments are added to each TESLA header.
- For each hash in the immediate authentication, the sender defines the maximum distance in intervals that the hash value spans. Usually, this distance is equal to the maximum key disclosure delay.

The sender needs to define the parameters of the cryptographic primitives that TESLA uses. The following parameters MUST be defined:

- The number n_1 of bits used for the key chain keys.
- A function F that is used to compute the key chain. Recall that K_i = F(K_{i+1}). Hence F MUST take an input of size n_1 and map it to an output of equal length.

- The number n_2 of bits used for the MAC key.
- A function F' to derive the MAC key from key chain keys. Recall that K'_i = F'(K_i). F' takes an input of n_1 bits and outputs n_2 bits.
- A MAC function that takes a key of length n_2 bits and data of any length; and outputs a MAC of length n_3.
- A hash function H that is used for the immediate authentication. H takes an input of arbitrary length and outputs n_4 bits.

Appendix A presents a list of the TESLA attributes.

For each of these values, n_i needs to be smaller or equal length than the function that generates the parameter. If the required output is smaller, we truncate is as specified in RFC 2104 [25].

If the sender and the receiver are already time synchronized, the sender may periodically broadcast signed packets that contain the above interval and key chain information. A new receiver would simply wait for such a packet, authenticate it with traditional schemes, and it can authenticate subsequent packets of the stream using TESLA. This is particularly useful in the case of satellite broadcast, since the receiver does not need to send any messages to the sender to bootstrap the authentication.

5.3 Sending Authenticated Packets

Each key is used in one time interval. However many messages are sent in each interval I_j, the key K_{j+d} is used to compute the MAC of all those messages. This allows the sender to send packets at any rate and to switch the sending rate dynamically. The key K_{j+d} remains secret for d-1 future intervals and is disclosed in interval I_{j+d}.. Packets sent in interval I_j can hence disclose key K_j. As soon as the receivers receive that key, they can verify the authenticity of the packets sent in interval I_{j-d}. Figure 2 shows the time intervals and the arrows point from the interval to the interval of which the key is disclosed. In the figure, d=2, hence packet P_2 sent in interval I_{j+3} discloses key K_{j+3}. From this key, the receiver can also recover K_{j+2} and verify the MAC of P_1.

5.4 Receiver Tasks

Since the security of TESLA depends on keys that remain secret until a pre-determined time period, the receiver MUST verify for each packet that the key, which is used to compute the MAC of that packet,

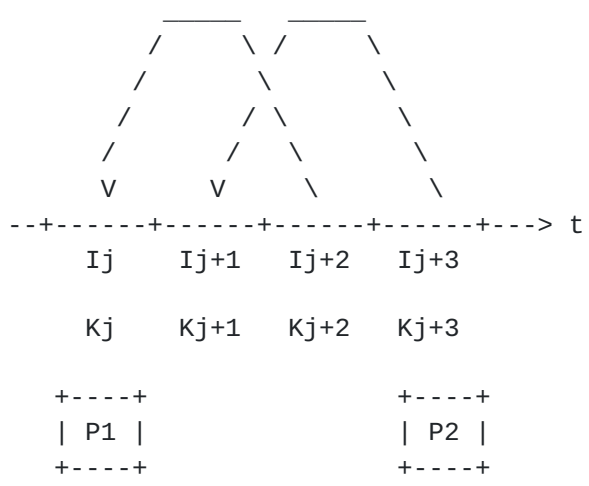

Figure 2: TESLA intervals and key disclosure

is not yet published by the sender. Otherwise an attacker could have changed the message data and re-computed the MAC. This motivates the security condition, which the receiver MUST verify for each packet it receives.

Security condition: A packet arrived safely, if the receiver is assured (based on its synchronized time and d_t) that the sender is not yet in the time interval in which the corresponding key is disclosed.

The formal definition of the security condition is in section 7.1. The intuition is that no attacker could have altered the packet in transit, because the corresponding MAC key is not yet disclosed by the sender. In case the security condition is not valid, the receiver MUST drop that packet, because the authenticity is not assured any more.

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 propagation delay.

We can now explain how the authentication with TESLA works with a concrete example. Figure 2 shows an example of two packets sent in different intervals. When the receiver receives packet P_i sent in interval I_j at time t_i (meaning the time the entire packet is received), it first evaluates the security condition. For this, the receiver computes the highest interval the sender could possibly be

in, in this case that interval is x = (t_i - TI_0 + d_t) / Tint. It now needs to verify x < I_j + d (I_j is the interval index) which means that the sender cannot be in the interval when the key K_j is disclosed, hence no attacker can possibly know that key and spoof the message contents.

Another property (could be called sanity condition) is that the receiver can verify if the interval id I_j is possible and is not in the future, i.e. if the sender can already be in that interval. The verification condition is that the following MUST hold I_j < (t_i - TI_0 + d_t) / Tint. This test prevents a denial-of-service attack described in more detail in section 7.3.

The receiver cannot, however, verify the authenticity of the message yet. Instead, it stores the triplet (I_j, M_i, MAC(K'_j, M_i)) to verify the authenticity later when it knows K'_j. Two possibilities exist on how to handle the unauthenticated message chunk M_i. The first possibility is to hand M_i to the application, and notify it through a callback mechanism as soon as M_i is verified. The second possibility it to keep M_i until the authenticity can be checked and pass it to the application as soon as M_i is authenticated.

If the packet contains a disclosed key, regardless of whether the security condition is verified or not, the receiver checks whether it can use K_{j-d} to authenticate previous packets. Clearly, if it has received K_{j-d} previously, it does not have any work to do. Otherwise, let us assume that the last key value in the reconstructed key chain is K_{v}. The receiver verifies if K_{j-d} is legitimate by applying the pseudo-random function F (j - d - v) -times and by verifying that the result is indeed K_{v}. More formally, K_{v} == F^{j-d-v}(K_{j-d}). If that condition is correct, the receiver updates the key chain. For each new keys K_{w}, it computes K'_{w} = F'(K_w) which might allow it to verify the authenticity of previously received packets.

It is clear that this system can tolerate arbitrary packet loss, because the receiver can verify the authenticity of all received packets eventually.

5.5 Multiple Authentication Instances

The key disclosure period in TESLA presents a tradeoff. 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 instances with different disclosure periods simultaneously. Each receiver can then use the instance 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 instance C_i, and drops the packet if none of the conditions are satisfied. Assume that the sender uses n authentication instances, where the first instance has the smallest delay until the disclosure packet is sent, and the nth instance has the longest delay. Furthermore, assume that for the incoming packet P_j, the security conditions for instances C_v (v < m) are not satisfied, and the condition for instance C_m is satisfied. In this case, as long as the key disclosure packets for the instances 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 instance C_v (v >= m) arrives, the receiver is assured of the authenticity of the packet P_j.

Instead of using one independent key chain per TESLA instance, we use the same key chain and time interval for all instances, but each instance has a different key disclosure delay. Each key K_i in the key chain is associated with the corresponding time interval T_i, and K_i will be disclosed in T_i. Assume that the sender uses n instances of TESLA, which we denote with A_1 ... A_n. Each TESLA instance A_u has a different key disclosure delay d_u, and it will have a MAC key schedule derived from the key schedule shifted by d_u time intervals from the key disclosure schedule. Let K^u_{i+d_u} denote the MAC key used by instance u in time interval T_i. We derive K^u_{i+d_u} as K^u_{i+d_u} = MAC(K_{i+d_u},u). The reason for generating all different, independent keys for each instance is to prevent an attack where an attacker moves the MAC value of an instance to another instance, which might allow it to claim that data was sent in a different interval. Our approach of generating independent keys prevents this attack. Thus to compute the MAC value in packet P_j in time interval T_i, the sender computes one MAC value of the message chunk M_j per instance and append the MAC values to M_j. In particular, for the instance A_u with disclosure delay d_u, the sender will now use the key K^u_{i+d_u} as mentioned above for the MAC computation.

Figure 3 shows an example with two TESLA instances, one with a key disclosure time of two intervals and the other of four intervals. The lowest line of keys shows the key disclosure schedule, i.e. which key is disclosed in which time interval. The middle and top line of keys shows the key schedule of the first and second instance respectively, i.e. which key is used to compute the MAC for the packets in the

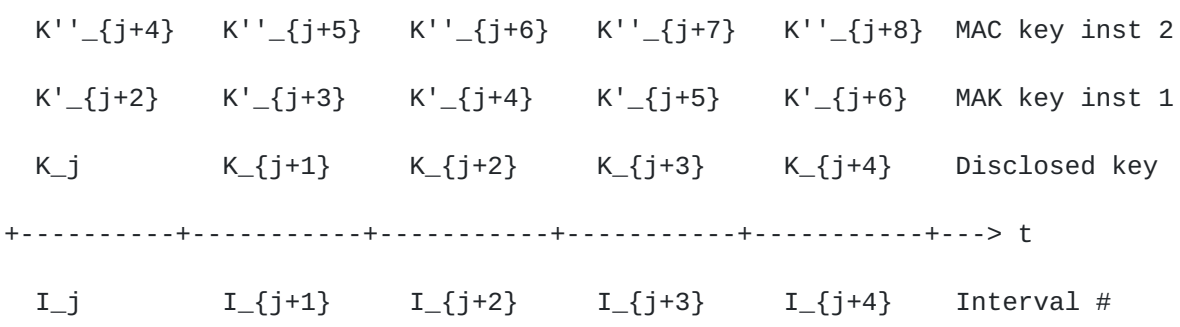

Figure 3: TESLA intervals and key disclosure

given time interval for the given instance. Using this technique, the sender will only need to disclose one key no matter how many instances are used concurrently.

5.6 Immediate Authentication

A drawback of the basic TESLA protocol is that the receiver needs to buffer packets during one disclosure delay before it can authenticate them. This might not be practical for certain applications if the receivers cannot afford much buffer space and bursty traffic might cause the receivers to drop packets due to insufficient buffer space. Moreover, as we show later in section 7.3, the requirement of receiver buffering introduces a vulnerability to a denial-of-service attack. To solve these problems caused by receiver-buffering, we propose a new method to support immediate authentication, which allows the receiver to authenticate packets as soon as they arrive.

The basic observation of this method is that we can replace receiver buffering with sender buffering. If the sender can buffer packets during one disclosure delay, then it could store the hash value of the data of a later packet in an earlier packet and hence as soon as the earlier packet is authenticated, the data in the later packet is authenticated through the hash value as well.

In the new scheme, the sender buffers packets for the duration of one disclosure delay. For simplicity of illustration, we assume that the sender sends out a constant number v of packets per time interval. To construct the packet for the message chunk M_j in time interval T_i, the sender appends the hash value of the message chunk M_{j+vd} to M_j and then computes the MAC value also over H(M_{j+vd}) with the key K_i. Figure 4 illustrates how the packet P_j is constructed by

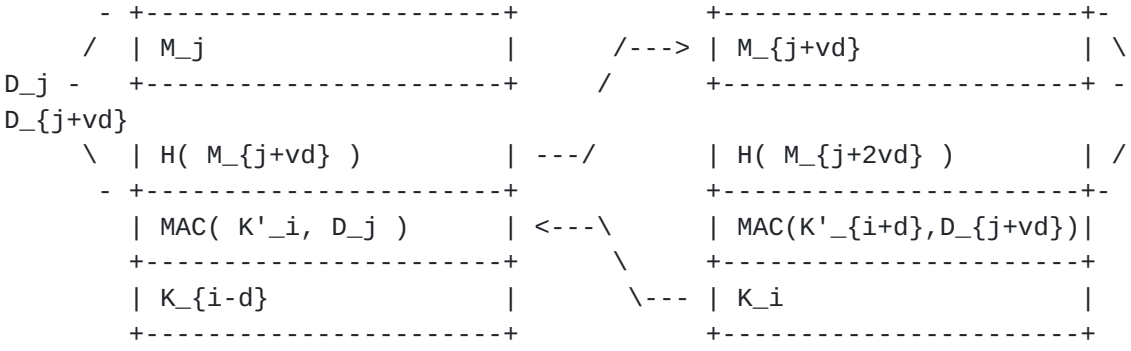

Figure 4: Immediate authentication packet example. D_j = H(M_{j+vd}) | M_j and D_{j+vd} = H(M_{j+2vd}) | M_{j+vd}.

appending H(M_{j+vd}), MAC(K_i, M_j | H(M_{j+vd})), along with the disclosed key K_{i-d}. (Note that the | stands for message concatenation). When the packet P_{j+vd} arrives at the receiver which discloses the key K_i it allows authentication of packet P_j sent in interval I_i. P_j carries a hash of the data M_{j+vd} in P_{j+vd}. If P_j is authentic, H(M_{j+vd}) is also authentic and therefore the data M_{j+vd} is immediately authenticated. Also note that if P_j is lost or dropped due to violation of the security condition, P_{j+vd} will not be immediately authenticated and can still be authenticated later using the MAC value. To achieve higher robustness to lost packets, the sender may add multiple hashes of future packets to the data packet, as we show in the data format in the following section.

The above description assumes a constant sending rate, an assumption which might not be the satisfied in practice. If the sending rate varies, matching the hash to the data with simple counting does not work any more. A simple solution would be to add the sequence number of the packet that contains the data to the hash. Unfortunately, this introduces overhead. A simple trick can save space in the TESLA header. From the connection setup, the receiver knows for each hash the interval in which the data will be sent in. Hence, to match the hash to the data, the receiver stores for each future interval the authenticated hashes that it received. For each incoming packet, the receiver can compute the hash of the data and search if the corresponding hash is present, which would immediately authenticate the data.

5.7 TESLA Authentication Field Format for MESP

TESLA can be used to authenticate the content of a variety of protocols. We discuss the format of the TESLA authentication field for the

case where it is used as the internal or external authentication scheme in MESP [17]. The MESP (for Multicast ESP) header is proposed as the header for authenticated and/or encrypted multicast traffic.

The approach of MESP for designing the packet format is to use a connection setup phase in which the various parameters of the header field are defined (such as authentication type and field lengths). Hence, subsequent headers fields do not contain type or length information.

Figure 5 shows the authentication data field format for authenticating message M_i. The fields have the following semantics. Note that if the length of each field will be rounded up to the next byte boundary.

- Disclosed key present flag (D): 1 bit
  D=1 indicates that this packet discloses the key of a previous interval. If D=0, the disclosed key is not present.

- Commitment to new key chain present flag (C): 1 bit
  C=1 indicates that the commitment value of the next key chain is present, only used shortly before switching to a new key chain. The common case is that C=0 and the key is not present. Section 6.4 has more details on how to switch key chains.

- The last key from the previous key chain disclosed again flag (L): 1 bit
  L=1 indicates that the last key of the previous key chain is disclosed again. The common case is L=0 and the disclosed key is not present. Section 6.4 has more details on how to switch key chains.
  NOTE: Since the size of the authentication field in MESP is fixed, exactly one of these fields MUST be set to one (and therefore one of these fields MUST be present).

- Reserved (Res): 5 bits
  All these bits MUST be set to 0.

- Disclosed key: n_1 bits if present
  Disclosing the interval key is OPTIONAL, since any key can be derived from a later interval key. The implementor needs to be careful though since a sparse key disclosure can potentially lead to an increased authentication delay.

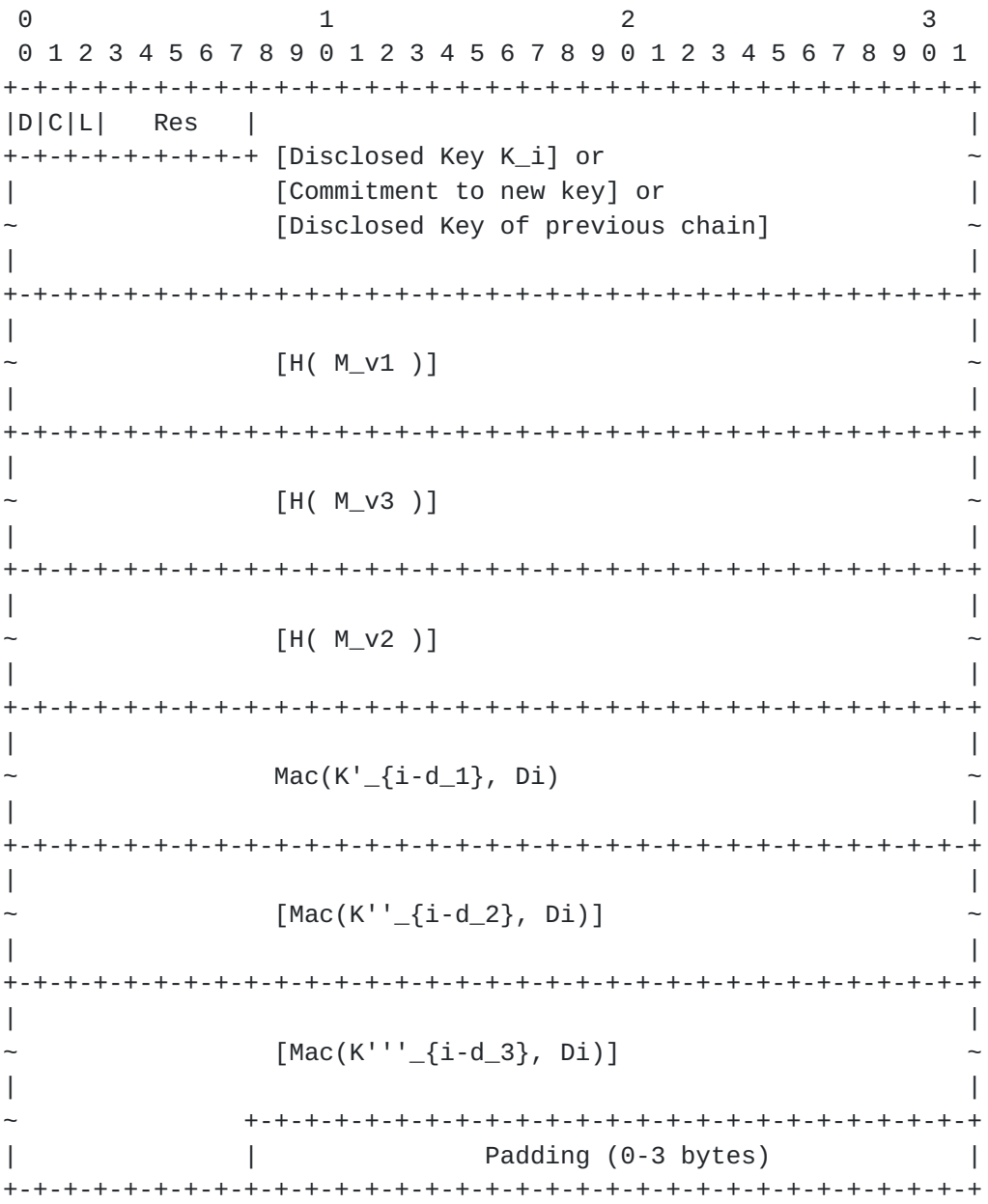

Figure 5: TESLA authentication field format in MESP

- Commitment to new key chain: n_1 bits if present
  This field is OPTIONAL. When the key chain tapers off and the sender wants to continue sending, the key chain needs to be changed and a commitment to the new key chain is added to the packet. It is implicit that the new chain will have the same length as the old key chain. A more detailed discussion is in

section 6.4.

- Disclosed key from previous chain: n_2 bits if present
  This field is OPTIONAL. Since the last key from the previous chain cannot be recovered from the current keys, a receiver might not have received the last key of a changed key chain, so it might need to be retransmitted. See section 6.4 for details.

- H(M_vi): n_4 bits
  These fields are OPTIONAL. If immediate authentication is used, the hashes of future data packets are present. From the connection setup, the receiver knows for each hash the interval in which the data will be sent in. Hence, to match the hash to the data, the receiver stores for each future interval the authenticated hashes that it received. For each incoming packet, the receiver can search if the corresponding hash is present, which would immediately authenticate the data.

- MAC: n_3 bits
  The MAC MUST be present in the packet. The MAC is computed over all data that needs authentication, which we denote with D_i, so MAC (K'_j, D_i). The MESP draft defines the data that is in D_i for both the internal and external authentication the fields that are included in the MAC. An exception, however, is that if the TESLA header contains a commitment to a new key of a new key chain, that key MUST be authenticated, hence it MUST be included in the data part of the computation of the MAC. Another exception are the hash value of the immediate authentication, which MUST also be authenticated, and hence included in the MAC computation. The format of the MAC computation is
  MAC(K'_j, D_i [| K_{new chain}] [| h_1 | ... | h_m]).
  As mentioned, D_i corresponds to the data that needs to be authenticated as described in [17]. If the commitment to the new key chain is in the packet, it is included in the MAC computation, as well as the hashes of future data, if immediate authentication is used.

- Padding: 0-3 bytes
  Pads the authentication block to the next 4 byte boundary. The padding field MUST be set to 0.

Note that no sequence number is present in the MESP version of the TESLA header, because MESP already features sequence numbers. The purpose of the sequence number is to filter out duplicates. Since TESLA does not provide duplicate protection, the packet sequence number helps the TESLA implementation to filter out duplicate messages for the receiver. Another method to remove duplicates is to use the

MAC of the message and remove messages that were sent in the same interval and have identical MAC's.

Also, the header we present does not include an explicit interval number. However, the receiver can infer the interval from the disclosed key. For reasonable implementations, changing key chains is extremely rare, and hence almost all packets will contain the disclosed key that implicitly informs the receiver of the time interval. Even if the disclosed key is not present, the receiver may infer the time interval by a brute-force verification of which time interval the packet was sent in.

5.8 TESLA for Authentication in other Protocols

Other authentication fields in other protocols may require different formats for the TESLA authentication field. For example, the Asynchronous Layer Coding (ALC) draft of the Reliable Multicast Group (RMT) also has an authentication field in their header, which needs a length and type field inside the authentication information [19]. It is straightforward to adapt TESLA for this format.

It is also possible to use the AMESP format with TESLA authentication (as described in Section 5.7) for the ALC authentication filed. See more details in [17].

5.9 Security Discussion

We briefly discuss potential security problems in this section and show how we address them in TESLA. First, we note that a more rigorous proof of security is detailed in [8].

The main point is that an attacker can perform at most a denial-of-service attack. Since we assume that the attacker has full control over the network, a denial-of-service attack is always possible (the attacker can simply drop or alter all packets). In the proof in [8] we show that an attacker cannot forge a MAC or invert the key chain, otherwise the attacker can break or invert the MAC function, which we assume is not possible. Similarly, the attacker cannot alter the packets undetectable, because she could not recompute the cryptographically secure MAC. Consequently, creating new packets is not possible either.

The security of TESLA is clearly compromised if an attacker could alter the receiver's clock. Hence, we assume that this is not possible. However, how is the security of TESLA influenced by speeding up or delaying individual packets? Delaying packets does not help, since the worst that could happen is for the receiver to drop the packet because it does not satisfy the security condition. Delaying

packets therefore results in a denial-of-service attack. On the other hand, speeding up packets does not do anything at all. The receiver even benefits from this since she might be able to use a chain with a short disclosure delay that she could not use otherwise.

The packet replay attack is prevented by adding a sequence number to each packet. Other measures against replay attacks are presented in section 5.7. The TESLA layer in the network stack or in the application will filter out duplicate packets. In any case, a duplication would be possible only within a short time period, since the security condition would cause packets to be dropped if they were duplicated with a long delay.

6 Implementation Considerations for the Sender

This section discusses general implementation issues for the sender. This discussion will assume that the sender uses one authentication chain only.

6.1 Choosing the disclosure delay

The choice of the disclosure delay presents a tradeoff. If the disclosure delay 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 address this tradeoff.

The first approach is to use multiple authentication chains, to accommodate the heterogeneity of the receiver delays, described in section 5.5. The problem remains how to choose the disclosure delay.

The main requirement for the disclosure delay is that the majority of packets will validate the security condition. Suppose that an upper bound on the dispersion of any client is d_t (maximum clock offset) and the propagation delay of any (reasonable) client is at most d_p. Usually the dispersion is also dependent on the propagation delay, in practice d_t ~ RTT. For each packet, the receiver needs to verify that the corresponding key was not disclosed yet at the moment the packet arrives. Therefore, the disclosure delay D_d needs to be longer than the sum of the dispersion and the propagation delay D_d > d_t + d_p.

6.2 Choosing the interval duration

The interval duration affects other parameters of TESLA. One key is necessary for each time interval, and due to the one-wayness of pseudo-random functions, the sender needs to pre-compute the key chain before sending packets. Consequently, a short interval duration and a long expected sending time will result in a longer pre-computation time.

Another factor to consider for the interval duration is the accuracy of the security condition. Because the time is sampled into intervals, the security condition is evaluated using the interval as unit. The finer-grained (or shorter) the interval duration is, the easier it is to approximate the ideal disclosure duration. A simple example illustrates this. The disclosure delay always needs to be longer than one interval, because otherwise (assuming that the disclosure delay is only one interval) packets that are sent close to an interval boundary would always violate the security condition. To achieve at least two intervals for the disclosure delay, the interval duration should be shorter than half of the desired disclosure delay Tint < D_d / 2.

The security is also influenced by the choice of the interval duration. The same key is used to compute the MAC of all packets sent in that interval. If a large number of packets is sent in each interval, an attacker can gather the packet-MAC pairs to attack the system. So far, we are not aware of such an attack, but it is nevertheless prudent to minimize the number of MAC computations that use an identical key.

6.3 Selecting a PRF and a MAC

TESLA requires two pseudo-random functions, one to generate the key chain and the other one to derive the MAC key from the key chain key. Figure 1 shows both functions. A good choice is to use HMAC for this purpose [25]. Any cryptographically secure hash function can be used in conjunction with HMAC, for example MD5 [26], SHA-1 [27], or RIPEMD-160. We propose the following constructs for our functions: F(x) = HMAC(x, 0) and F'(x) = HMAC(x, 1). Note that the first parameter to HMAC is the key and the second parameter is the data. For TESLA, the key size is 80 bits and the parameters 0 and 1 are passed as one-byte character (0x00 and 0x01 respectively). The output of this function is truncated to 80 bits, we pick the 80 most significant bits out of the 128 output bits.

Clearly, we also use HMAC for our MAC function. We simply have MAC(k,d) = HMAC(k,d) and we use the 80 most significant bits.

6.4 Altering key chains on the fly

If a sender distributes the same information for a long time, the pre-computed key chain can taper off. The question is how the sender can seamlessly switch to a new key chain. A simplistic approach is for the sender to stop using the old chain, send a new authenticated packet that commits to the new chain and to subsequently use the new chain only. The shortcoming of this scheme is that a receiver might not receive the authenticated packet and could not authenticate subsequent packets. Another approach is to use two key chains that overlap during a short transition period. The shortcoming of this approach is that the authentication field size would need to grow larger during the transition period, which conflicts with the fixed authentication field size in some standards mentioned previously.

A better approach is to commit to the new key chain by using previous TESLA packets. The challenge of packet loss though remains. A solution is to add the commitment to the new key chain to a number of packets. The sender needs to start sending the new key chain commitment value early enough, such that the majority of receivers will receive and authenticate the new key chain before the previous chain expires. This ensures that the switching to the new chain happens seamlessly.

Figure 6 shows an example. One key chain stops at Key K_n and the next chain starts with key K_{n+1}. The sender would then include a commitment to the new key chain into multiple packets of the previous chain, so F(K_{n+1}) would be added to packets throughout intervals I_{n-3} through I_n. Since the keys of the new chain are first revealed in interval I_{n+4}, it is sufficient if the commitment to the new key chain is sent in interval I_n (because the key K_n is disclosed in interval I_{n+3}). Again, to guard against packet loss it is safer to add the commitment to multiple packets before interval I_n. Similarly, the key for interval I_n is disclosed by packets sent in interval I_{n+3}, but if interval I_{n+3} packets are either not sent or lost, the receiver could not verify the packets from interval I_n. To guard against this event, the sender would include K_n in a packets sent after interval I_{n+3}.

7 Implementation Considerations for the Receiver

This section discusses implementation considerations for the receiver.

7.1 Time Synchronization and Security Condition Issues

We assume that the time at the receiver has negligible drift. In case the time drift can be substantial, periodic time synchronization is necessary. In appendix B, we propose two time synchronization schemes

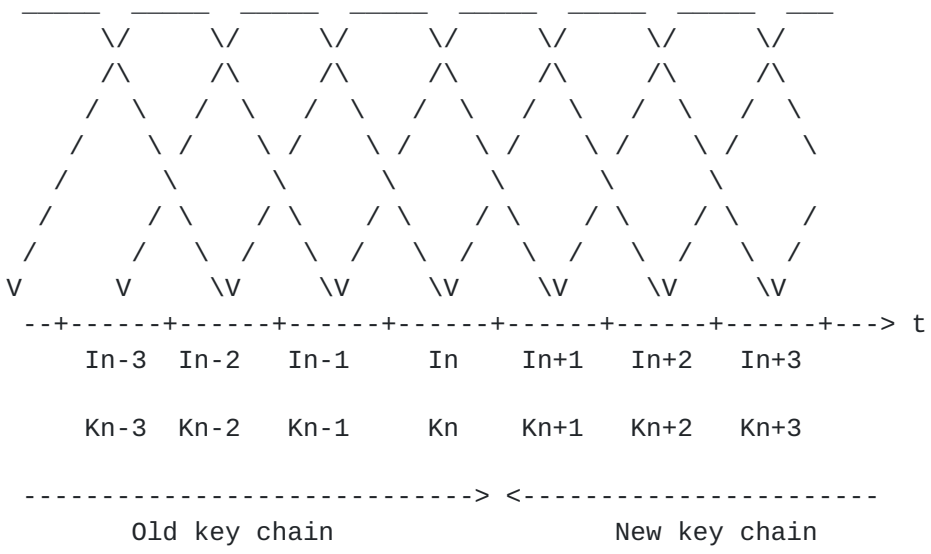

Figure 6: Key chain change

for TESLA. Either the receiver synchronizes its time with the sender directly, or indirectly with a time synchronization server in the network. In the latter case, the receiver would need to wait for an authenticated interval and key chain announcement packet, before it can start authenticating packets of the data stream. Even if the receiver is not time synchronized, it can already receive packets and record their arrival times. Later in time after the time synchronization, the receiver can then start to evaluate their security condition and authenticate the packets.

Before sending the packet, the receiver records the current time T_s. When it receives the sender's response, the receiver immediately records the current time T_r. After this, the receiver verifies the digital signature using the public key of the server, and checks that the returned nonce is equal to the nonce sent (if the return packet contains the nodes of a hash tree, the receiver computes the root node of the hash tree and checks if that matches the value in the packet). If the check is unsuccessful or if the round-trip-time (RTT = T_r - T_s) passes a certain threshold, the synchronization needs to be repeated. The packet will contain the current time at the sender T_c.

During the protocol, the receiver needs to evaluate the earliest and the latest time that the sender can possibly be in. The latest possible time t_l at the server is t_l = t - T_s + T_c, where t is the current receiver time. Similarly, the earliest possible time at the sender is t_e = t - T_r + T_c.

To evaluate the security condition, the receiver computes the latest possible time that the sender can be at and it verifies that the sender is not yet in the time interval in which the corresponding key is disclosed. Assume that the packet arrival time is t_p and the interval is I_j. The corresponding key will be disclosed in interval I_{j+d}. The latest possible interval the server can be in is I' = integer((t_l - T_i) / Tint) + I_i, where T_i is the starting time of interval I_i. So the security condition is valid if I' < I_{j+d}.

Similarly, the receiver can ensure that the sender can even be in that interval, based on the earliest time the server can be in. This test prevents the denial-of-service attack outlined in section 7.3. The check is integer((t_e - T_i) / Tint) + I_i >= I_j.

7.2 Reconstruction of the key chain

The receiver reconstructs the key chain as time progresses. For each new key chain value it receives, it verifies the correctness of the key by calling the pseudo-random function F until it matches the last authenticated key chain value. Fortunately, it makes no sense for the receiver to store the entire key chain. As soon as a key value is used to authenticate packets in the packet buffer, no new packet could arrive using that same key since it would clearly violate the security condition. Hence a key can be immediately discarded after the corresponding packets were authenticated. Only the most recently disclosed key is used subsequently, because it is a commitment for the later keys in the key chain and it is therefore used to verify a new key.

7.3 Protecting against Denial-of-Service Attacks

A possible denial-of-service attack on the receiver would be as follows. A malicious attacker can send a packet with a sending interval that is far out in the future. When the receiver would try to verify the disclosed key, it would need to perform a large number of pseudo-random function computations, which would cause it to waste large amounts of its computation resources. Fortunately, this attack is easy to prevent. Analogous to the security condition, the receiver verifies for each packet whether it is even possible that the sender has sent that particular packet. If this check is not satisfied, the packet must be spoofed and the receiver immediately drops the packet.

8 Acknowledgments

We would like to thank Mike Luby for his feedback and support.

9 Bibliography

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

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

[3] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas, "Multicast security: A taxonomy and some efficient constructions," in Infocom '99 , 1999.

[4] R. Gennaro and P. Rohatgi, "How to Sign Digital Streams," tech. rep., IBM T.J.Watson Research Center, 1997.

[5] P. Rohatgi, "A compact and fast hybrid signature scheme for multicast packet authentication," in 6th ACM Conference on Computer and Communications Security , November 1999.

[6] P. Rohatgi, "A hybrid signature scheme for multicast source authentication," Internet Draft, Internet Engineering Task Force, June 1999. Work in progress.

[7] C. K. Wong and S. S. Lam, "Digital signatures for flows and multicasts," in Proc. IEEE ICNP `98 , 1998.

[8] A. Perrig, R. Canetti, J. Tygar, and D. X. Song, "Efficient authentication and signing of multicast streams over lossy channels," in IEEE Symposium on Security and Privacy , May 2000.

[9] A. Perrig, R. Canetti, D. X. Song, and J. Tygar, "Efficient and secure source authentication for multicast," in Network and Distributed Systems Security NDSS 2001 , 2001.

[10] B. Briscoe, "Flames: Fast, loss-tolerant authentication of multicast streams," tech. rep., BT Research, 2000. http://www.labs.bt.com/people/briscorj/papers.html.

[11] F. Bergadano, D. Cavagnino, and B. Crispo, "Chained stream authentication," in Selected Areas in Cryptography 2000 , (Waterloo, Canada), August 2000. A talk describing this scheme was given at IBM Watson in August 1998.

[12] S. Cheung, "An efficient message authentication scheme for link state routing," in 13th Annual Computer Security Applications Conference , 1997.

[13] F. Bergadano, D. Cavalino, and B. Crispo, "Individual single source authentication on the mbone," in ICME 2000 , Aug 2000. A talk containing this work was given at IBM Watson, August 1998.

[14] S. Bradner, "Key words for use in RFCs to indicate requirement levels," Request for Comments (Best Current Practice) 2119, Internet Engineering Task Force, Mar. 1997.

[15] Secure Multicast Group (SMuG). http://www.ipmulticast.com/community/smug/.

[16] R. Canetti, P. Cheng, D. Pendarakis, J. Rao, P. Rohatgi, and D. Saha, "An architecture for secure internet multicast," Internet Draft, Internet Engineering Task Force, Feb. 1999. Work in progress.

[17] R. Canetti, P. Rohatgi, and P.-C. Cheng, "Multicast data security transformations: Requirements, considerations, and prominent choices," internet draft, Internet Engineering Task Force, 2000. draft-data-transforms.txt.

[18] Reliable Multicast Transport (RMT). http://www.ietf.org/html.charters/rmt-charter.html.

[19] M. Luby, J. Gemmell, L. Vicisano, L. Rizzo, J. Crowcroft, and B. Lueckenhoff, "Asynchronous layered coding. a massively scalable reliable multicast protocol," internet draft, Internet Engineering Task Force, July 2000. draft-ietf-rmt-pi-alc-01.txt.

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

[21] L. Lamport, "Password authentication with insecure communication," Communications ACM , vol. 24, Nov. 1981.

[22] N. Haller, "The S/KEY one-time password system," Request for Comments (Informational) 1760, Internet Engineering Task Force, Feb. 1995.

[23] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," Communications ACM , vol. 21, no. 2, pp. 120--126, 1978.

[24] "Digital Signature Standard (DSS), Federal Register 56." FIPS PUB 186, Aug. 1991.

[25] M. Bellare, R. Canetti, and H. Krawczyk, "HMAC: Keyed-hashing for message authentication," Internet Request for Comment RFC 2104, Internet Engineering Task Force, Feb. 1997.

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

[27] C. S. Laboratory), "Secure hash standard." Federal Information Processing Standards Publication FIPS PUB 180-1. http://csrc.nist.gov/fips/fip180-1.txt (ascii), http://csrc.nist.gov/fips/fip180-1.ps (postscript), Apr. 1995.

[28] M. Baugher, T. Hardjono, and B. Weis, "Group domain of interpretation for isakmp," internet draft, Internet Engineering Task Force, July 2000. draft-irtf-smug-gdoi-00.txt.

[29] M. Bishop, "A Security Analysis of the NTP Protocol Version 2," in Sixth Annual Computer Security Applications Conference , November 1990.

[30] R. Merkle, "Protocols for public key cryptosystems," in 1980 IEEE Symposium on Security and Privacy , 1980.

A TESLA Attributes

The following attributes need to be defined in the connection startup phase for TESLA. This list is extended from [28] and will be completed in the next version of this draft.

ID Class	Value
RESERVED	0
SOURCE_ID	64
DIRECT_SYNCHRONIZATION	65
SENDERS_CERT_TYPE	66
SENDERS_CERT	67
MAC_TYPE	68
MAC_LENGTH	69
KEY_CHAIN_PRF	70
KEY_CHAIN_KEY_LENGTH	71
INTERVAL_NUMBER	72
INTERVAL_STARTING_TIME	73
INTERVAL_DURATION	74
KEY_DISCLOSURE_DELAY	75

KEY_CHAIN_COMMITMENT_VALUE	76
KEY_CHAIN_EXPIRATION_VALUE	77
NUMBER_OF_TESLA_INSTANCES	78
IMMEDIATE_AUTH_HASH_NUMBER	79
IMMEDIATE_AUTH_HASH_TYPE	80
IMMEDIATE_AUTH_HASH_LENGTH	81

B Time Synchronization Packet Format

We present two options for synchronizing the time. Either the receiver directly synchronizes its time with the sender (direct synchronization), or both the sender and receiver synchronize their time with a third entity, a time server (indirect synchronization). We discuss both cases separately.

B.1 Direct Synchronization

In direct synchronization, each receiver synchronizes its time with the sender, and registers the dispersion (maximum clock offset between the sender and receiver clock). We remark that the receiver only needs to know a rough upper bound d_t on the dispersion (where d_t can be on the order of multiple seconds). (Many clock synchronization algorithms exist, for example the work of Mills on NTP [20], and its security analysis [29].) In section 7.1 we describe a simple protocol and discuss scalability issues related to the initial synchronization.

The receiver sends a time synchronization request to the sender. To ensure that the senders response is fresh and not replayed by an attacker, the receiver creates a nonce N_r and sends it to the sender. Figure 7 shows the request information.

The sender will then return this nonce in the response, which allows the receiver to ensure that the response packet is fresh. The response packet contains the following components:

- The current sender time
- The beginning time of a specific interval TI_j
- The id of that interval I_j
- The interval duration Tint

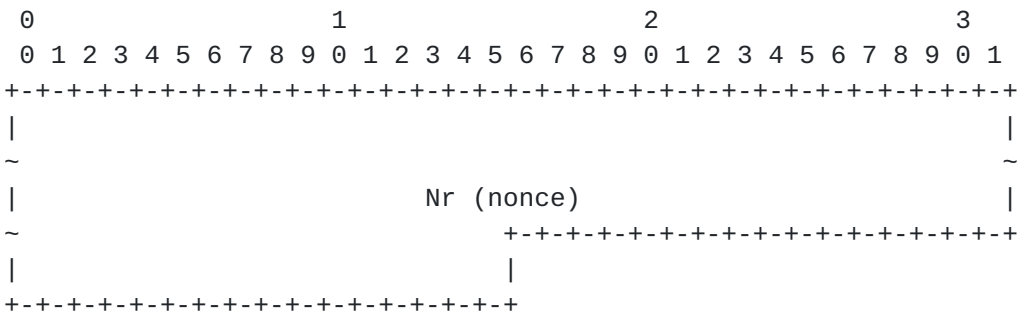

Figure 7: Time synchronization request packet format

- Key disclosure delay d (unit is interval)
- A publically known key K_i (i < j - d where j is the current interval index)
- The interval of the last key chain element (key chain expiration)
- Total key chain length
- The nonce N_r

Figure 8 shows the outline of the direct synchronization packet format. If the T-bit is set, multiple users requested synchronization concurrently and a hash tree was formed over the nonces. Hence the signed nonce is the root of the hash tree and the other nodes used for reconstruction is added outside of the signature at the end of the packet. If the bit is clear, no hash tree is added and the value in the nonce field is the users nonce.

The times are specified in milliseconds (ms). Absolute times are expressed in the number of milliseconds since January 1, 1970. Since we use 64-bit time fields, this value will roll over in the year 584944387 AD. The interval id's and lengths are in 32-bit fields. Finally, the packet is signed. The digital signature includes the bitfield and ends at the nonce.

The number of nodes specifies how many nodes of the hash tree follow. This number is followed by a list of 10-byte tree hash nodes, starting at the lowest node. The last hash node in the list a child node of the root node.

The receiver uses a nonce in its first packet to prevent a replay attack by an attacker that replays a previously signed synchronization reply. Besides the current sender time t_S, 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 digital signature scheme.

Since an adversary could artificially speed up either the sender's or the receiver's packet, the timestamp t_S that the sender returns could be either from the instant the request was sent or from the instant the request is received. The receiver sets its local time to t_S and sets d_t = RTT, where RTT stands for the round-trip time. The sender's time is assured to be within the interval [t-d_t,t+d_t], where t is the synchronized local time of the receiver. This synchronization is quite primitive, but it is secure against an active adversary and it's accuracy suffices for our application. To increase the accuracy, the receiver can run the protocol multiple times and use the run which has the lowest RTT. Any other time synchronization protocol can be used, as long as it is robust against an active adversary (most schemes do not satisfy this requirement). An example for a time synchronization protocol is Mills NTP [20].

This response packet needs to be authenticated with a digital signature scheme, such as RSA [23], or DSA [24]. We assume that a public-key infrastructure is in place (PKI). Since most public-key signature algorithms are computationally expensive, the signing of the response packet can become a performance bottleneck for the sender. (A 500 MHz Pentium III workstation can compute on the order of 100 RSA 1024-bit signatures per second.) A simple trick can alleviate this situation. The sender can aggregate multiple requests, compute and sign a Merkle hash tree that is generated from all the requester's nonces [30]. Figure 9 shows how such a hash tree is constructed. If N_h is the root of the hash tree, N_h would be included in the signed part of the response packet instead of N_r. To verify the digital signature of the response packet, each receiver would reconstruct the hash tree. Since it does not know the other receiver's nonces that are part of the hash tree, the sender would include the nodes of the tree necessary to reconstruct the root node. For the example in figure 9, the packet returned to receiver A would include N_b and H_{cd}. Receiver A can reconstruct the root node H_{ad} from these values and its own nonce N_a as follows: H_{ad} = H(H(N_a,N_b),H_{cd}). Note that the number of nodes returned in the response packet is logarithmic in the number of receivers whose request arrived in the same time interval. Assuming a 50 ms interval time (the sender would need to compute at most 20 signatures per second) and assuming that 1,000,000 receivers wanted to synchronize their time in that interval, the return packet would only need to contain 20 hash nodes or 200 bytes,

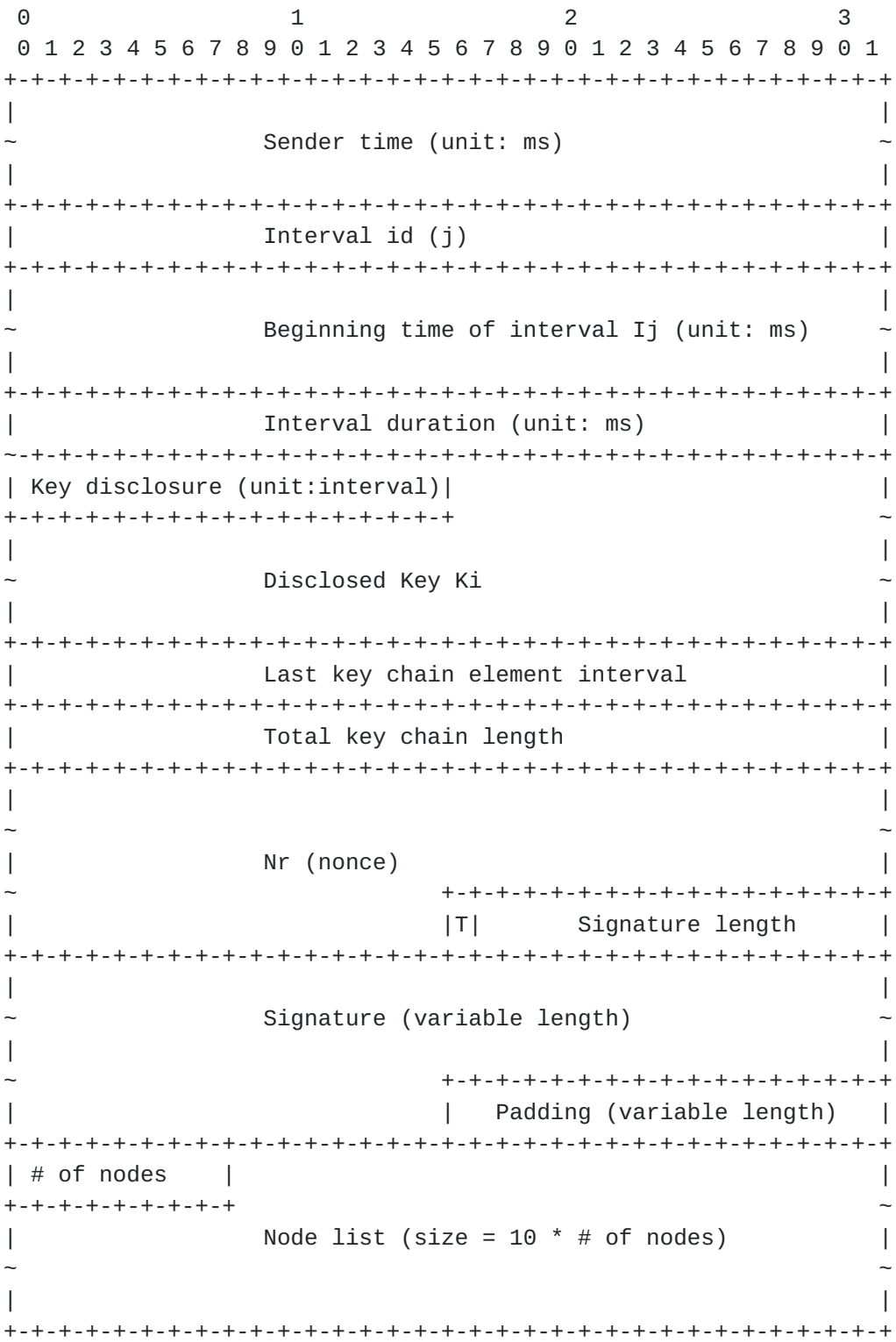

Figure 8: Direct synchronization packet format

assuming an 80 bit hash function. Any cryptographically secure hash function can be used for H(x,y), for example MD5 [26].

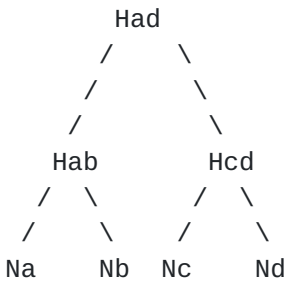

Figure 9: Hash tree over receiver nonces. Node H_{ab} = H(N_a, N_b). H_{ad} = H(H_{ab},H_{cd}).

B.2 Indirect Synchronization

If distributed time synchronization servers exist in the network, the sender and receivers can separately synchronize their clocks with the same time synchronization server. The resulting dispersion is the sum of their dispersions with the time synchronization server. NTP is a sophisticated time synchronization protocol that can be used in this case [20].

To bootstrap the authentication process, an initial authenticated packet is still required, which can be authenticated with a digital signature scheme. The sender periodically broadcasts an interval specification message, digitally signed with its private key. The initial authenticated packet contains the precise interval information (interval frequency and the starting time of a specific interval) along with the key of a past interval. The packet contains the following fields:

- The Id / IP address of the time synchronization server
- The dispersion to the time synchronization server
- The beginning time of a specific interval TI_j
- The id of that interval I_j
- The interval duration Tint

- Key disclosure delay d (unit is interval)
- A publically known key K_i (i < j - d where j is the current interval index)
- The interval of the last key chain element (key chain expiration)
- Total key chain length

The packet format is depicted in figure 10. The bitfield is currently unused in this version and reserved for future purposes. The remaining fields are analogous to the direct synchronization packet, except that the current time is replaced with the IP address of the time synchronization server and the dispersion of the sender's clock with respect to the time synchronization server.

The main advantage for indirect synchronization is that the sender does not need to process any requests from the receiver, hence this scheme achieves maximum scalability. This also dramatically improves the security of the sender, since it can be configured not to receive any packets.

C Author Contact Information

Adrian Perrig
SIMS - UC Berkeley / Digital Fountain
102 South Hall, 4600
Berkeley, CA 94720
US
perrig@cs.berkeley.edu

Ran Canetti
IBM Research
30 Saw Mill River Rd
Hawthorne, NY 10532
US
canetti@watson.ibm.com

Bob Briscoe
BT Research
B54/74, BT Labs
Martlesham Heath
Ipswich, IP5 3RE
UK
bob.briscoe@bt.com

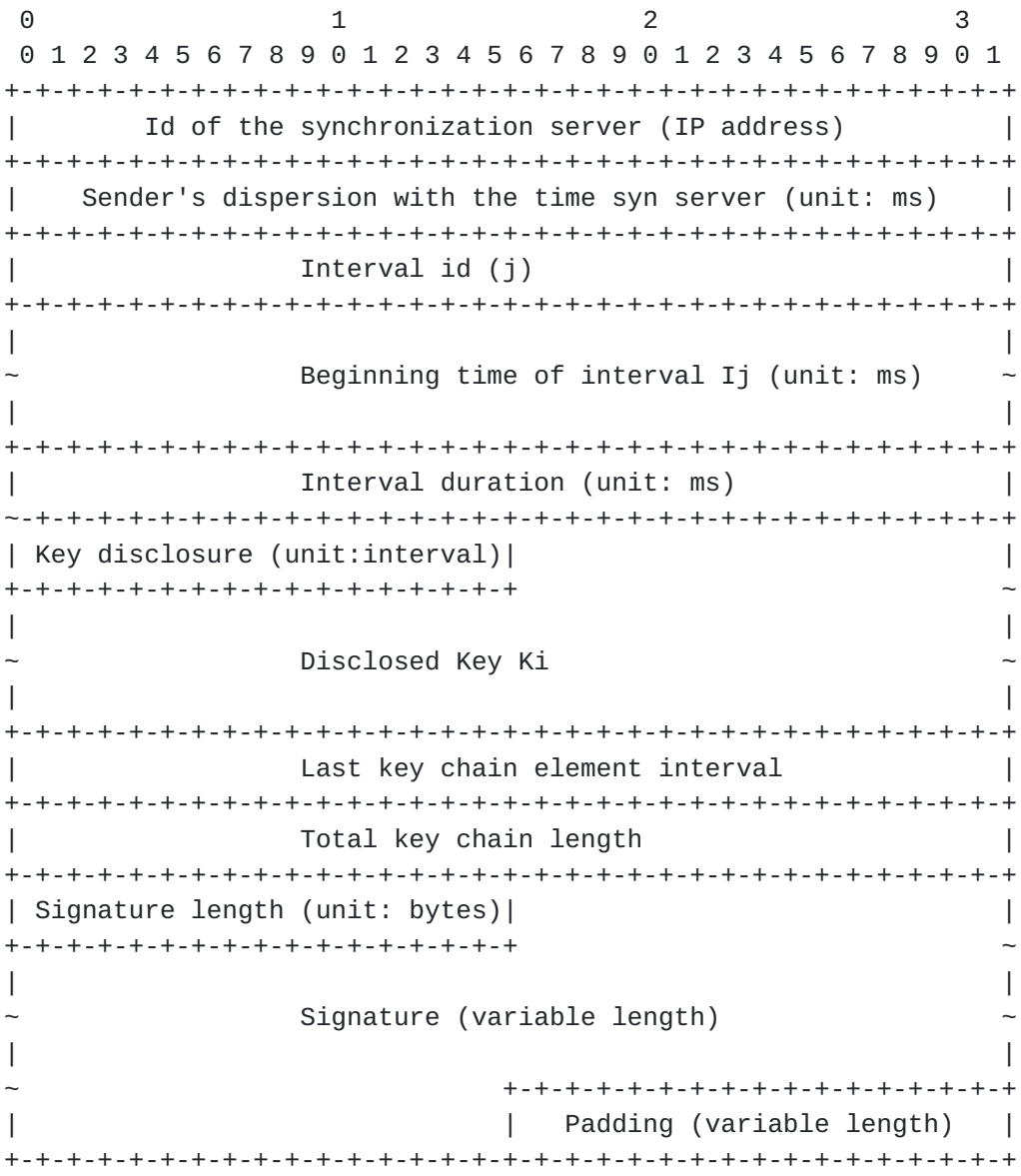

Figure 10: Indirect synchronization packet format

Doug Tygar
SIMS - UC Berkeley
102 South Hall, 4600
Berkeley, CA 94720-4600
US
tygar@cs.berkeley.edu

Dawn Song

CS - UC Berkeley
387 **Soda Hall, 1776**
Berkeley, CA 94720-1776
US
dawnsong@cs.berkeley.edu

D Full Copyright Statement