Internet Engineering Task Force
MSEC Working Group
INTERNET-DRAFT
Expires: June 2005

Perrig, Canetti, Song, Tygar, Briscoe
CMU / IBM / CMU / UC Berkeley / BT

December 2004

TESLA: Multicast Source Authentication Transform Introduction

<draft-ietf-msec-tesla-intro-04.txt>

STATUS OF THIS MEMO

Abstract

This document introduces TESLA, short for Timed Efficient Stream Loss-tolerant Authentication. TESLA allows all receivers to check the integrity and authenticate the source of each packet in multicast or broadcast data streams. TESLA requires no trust between receivers; uses low cost operations per packet at both sender and receiver; can tolerate any level of loss without retransmissions; and requires no per-receiver state at the sender. TESLA can protect receivers against denial of service attacks in certain circumstances. Each receiver must be loosely time synchronized with the source in order to verify messages, but otherwise receivers need send no messages. TESLA alone cannot support non-repudiation of the data source to third parties.

This informational document is intended to assist in writing standardizable and secure specifications for protocols based on TESLA in different contexts.

Table of Contents

1. Introduction

In multicast, a single packet can reach millions of receivers. This unfortunately also introduces the danger that an attacker can potentially also reach millions of receivers with a malicious packet. Through source authentication, receivers can ensure that a received multicast packet originates from the correct source. In these respects, a multicast is equivalent to a broadcast to a superset of the multicast receivers.

In unicast communication, we can achieve data authentication through a simple mechanism: the sender and the receiver share a secret key to compute a message authentication code (MAC) of all communicated data. When a message with a correct MAC arrives, the receiver is assured that the sender generated that message. Standard mechanisms achieve unicast authentication this way, for example TLS or IPsec [1,2].

Symmetric MAC authentication is not secure in a broadcast setting. Consider a sender that broadcasts authentic data to mutually

mistrusting receivers. The symmetric MAC is not secure: every receiver knows the MAC key, and hence could impersonate the sender and forge messages to other receivers. Intuitively, we need an asymmetric mechanism to achieve authenticated broadcast, such that every receiver can verify the authenticity of messages it receives, without being able to generate authentic messages. Achieving this in an efficient way is a challenging problem [3].

The standard approach to achieve such asymmetry for authentication is to use asymmetric cryptography, e.g., a digital signature. Digital signatures have the required asymmetric property: the sender generates the signature with its private key, and all receivers can verify the signature with the sender's public key, but a receiver with the public key alone cannot generate a digital signature for a new message. A digital signature provides non-repudiation, a stronger property than authentication. However, digital signatures have a high cost: they have a high computation overhead for both the sender and the receiver, and most signatures also have a high bandwidth overhead. Since we assume broadcast settings where the sender does not retransmit lost packets, and the receiver still wants to immediately authenticate each packet it receives, we would need to attach a digital signature to each message. Because of the high overhead of asymmetric cryptography, this approach would restrict us to low-rate streams, and to senders and receivers with powerful workstations. We can try to amortize one digital signature over multiple messages. However, such an approach is still expensive in contrast to symmetric cryptography, since symmetric cryptography is in general 3 to 5 orders of magnitude more efficient than asymmetric cryptography. In addition, the straight-forward amortization of one digital signature over multiple packets requires reliability, as the receiver needs to receive all packets to verify the signature. A number of schemes that follow this approach are [4,5,6,7,8]. See [9] for more details.

This document presents the Timed Efficient Stream Loss-tolerant Authentication protocol (TESLA). TESLA uses mainly symmetric cryptography, and uses time delayed key disclosure to achieve the required asymmetry property. However, TESLA requires loosely synchronized clocks between the sender and the receivers. See more details in Section 3.3.1. Other schemes that follow a similar approach to TESLA are [10,11,12].

1.1. 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., function F with the argument x

executed i times is F^i(x).

2. Functionality

TESLA provides delayed per-packet data authentication and integrity checking. The key idea to providing 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 for generation and verification of authentication information
- Low communication overhead
- Limited buffering required for the sender and the receiver, hence timely authentication for each individual packet
- Strong robustness to packet loss
- Scales to a large number of receivers
- Protects receivers against denial of service attacks in certain circumstances if configured appropriately
- Each receiver cannot verify message authenticity unless it is loosely time synchronized with the source, where synchronization can take place at session set-up. Once the session is in progress, receivers need not send any messages or acknowledgements.
- Non-repudiation is not supported - each receiver can know that a stream is from an authentic source, but not prove this to others later.

TESLA can be used either in the network layer, or in the transport layer, or in the application layer. Delayed authentication, however, requires buffering of packets until authentication is completed. Certain applications intolerant of delay may be willing to process packets in parallel to being buffered awaiting authentication, as long as roll-back is possible if packets are later found to be unauthenticated. For instance, an interactive video may play-out packets still awaiting authentication, but if they are later found to be unauthenticated, it could stop further play-out and warn the viewer that the last x msec were unauthenticated and should be ignored. However, in the remainder of this document, for brevity, we will assume packets are not processed in parallel to buffering.

2.1. 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, re-send, 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 find the key of a pseudo-random function (or distinguish it from a random function) with non-negligible probability.

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

Since TESLA expects the receiver to buffer packets before authentication, the receiver needs to protect itself from a potential denial-of-service (DOS) attack due to a flood of bogus packets (see section 3.8).

2.2. Assumptions

TESLA makes the following assumptions in order to provide security:

1. The sender and the receiver must be able to loosely synchronize. Specifically, each receiver must be able to compute an upper bound on the lag of the receiver clock relative to the sender clock. We denote this quantity by D_t. (That is, D_t = sender time - receiver time). We note that an upper bound on D_t can be easily obtained via a simple two-message exchange. (Such an exchange can be piggybacked on any secure session initiation protocol. Alternatively, standard protocols such as NTP [16] can be used.
2. TESLA MUST be bootstrapped at session set-up through a regular data authentication system. One option is 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). Alternatively,

this initialization step can be done using any secure session initiation protocol.

3. TESLA uses cryptographic MAC and PRF (pseudo-random functions). These MUST be cryptographically secure. Further details on the instantiation of the MAC and PRF are in Section 3.4.

4. We would like to emphasize that the security of TESLA does NOT rely on any assumptions on network propagation delay.

3. The Basic TESLA Protocol

TESLA is described in several academic publications: A book on broadcast security [13], a journal paper [14], and two conference papers [8,15]. Please refer to these publications for in-depth proofs of security, experimental results, etc.

We first outline the main ideas behind TESLA.

3.1. Protocol sketch

As we argue in the introduction, broadcast authentication requires a source of asymmetry. TESLA uses time for asymmetry. We first make sure that the sender and receivers are loosely time synchronized as described above. Next, the sender forms a one-way chain of keys, where each key in chain is associated with a time interval (say, a second). Here is the basic approach:

- The sender attaches a MAC to each packet. The MAC is computed over the contents of the packet. For each packet, the sender uses the current key from the one-way chain as a cryptographic key to compute the MAC.

- The sender discloses a key from the one-way chain after some pre-defined time delay. (e.g., the key used in time interval i is disclosed at time interval i+3.)

- Each receiver receives the packet. Each receiver knows the schedule for disclosing keys and, since it has an upper bound on the local time at the sender, it can check that the key used to compute the MAC was not yet disclosed by the sender. If so, then the receiver buffers the packet. Otherwise the packet is dropped due to inability to authenticate. Note that we do not know for sure whether a "late packet" is a bogus one or simply a delayed packet. We drop the packet since we are unable to authenticate it. (Of course, an implementation may choose to not drop packets and use them unauthenticated.)

- Each receiver checks that the disclosed key belongs to the hash-chain (by checking against previously released keys in the chain) and then checks the correctness of the MAC. If the MAC is correct, the receiver accepts the packet.

Note that one-way chains have the property that if intermediate values of the one-way chain are lost, they can be recomputed using subsequent values in the chain . So, even if some key disclosures are lost, a receiver can recover the corresponding keys and check the correctness of earlier packets.

We now describe the stages of the basic TESLA protocol in this order: sender setup, receiver bootstrap, sender transmission of authenticated broadcast messages, and receiver authentication of broadcast messages.

3.2. Sender Setup

The sender divides the time into uniform intervals of duration T_int. The sender assigns one key from the one-way chain to each time interval in sequence.

The sender determines the length N of the one-way chain K_0, K_1, ..., K_N, and this length limits the maximum transmission duration before a new one-way chain must be created. The sender picks a random value for K_N. Using a pseudo-random function (PRF) f, the sender constructs the one-way function F: F(k) = f_k(0). The rest of the chain is computed recursively using K_i = F(K_{i+1}). Note that this gives us K_i = F^{N-i}(K_N), so the receiver can compute any value in the key chain from K_N even if is does not have intermediate values. The key K_i will be used to authenticate packets sent in time interval i.

Jakobsson [21], and Coppersmith and Jakobsson [22] present a storage and computation efficient mechanism for one-way chains. For a chain of length N, storage is about log(N) elements, and the computation overhead to reconstruct each element is also about log(N).

The sender determines the duration of a time interval, T_int, and the key disclosure delay, d. (T_int is measured in time units, say milliseconds, and d is measured in number of time intervals. That is, a key that is used for time interval i will be disclosed in time interval i+d.) It is stressed that the scheme remains secure for any values of T_int and d. Still, correct choice of T_int and d is crucial for the usability of the scheme. The choice is influenced by the estimated network delay, the length of the transmission, and the tolerable delay at the receiver. T_int that is too short will cause the keys to run out too soon. T_int that is too long will cause excessive delay in authentication for some of the packets (those that were sent at the beginning of a time period). Delay d that is too short will cause too many packets to be unverifiable by the receiver. Delay d that is too long will cause excessive delay in authentication.

The sender estimates a reasonable upper bound on the network delay between the sender and any receiver as m milliseconds. This includes any delay expected in the stack (see section 4 on layer placement). If the sender expects to send a packet every n milliseconds, then a reasonable value for T_int is max(n,m). Based on T_int, a rule of thumb for determining the key disclosure delay, d, is given in section 3.6.

The above value for T_int is neither an upper or lower bound, merely the value that reduces key change processing to a minimum without causing authentication delay to be higher than necessary. So if the application can tolerate higher authentication delay then T_int can be made appropriately larger. Also, if m (or n) increases during the session, perhaps due to congestion or a late joiner on a high delay path, T_int need not be revised.

Finally, the sender needs to allow each receiver to synchronize its time with the sender. See more details on how this can be done in section 3.3.1. (It is stressed that estimating the network delay is a separate task than the time synchronization between the sender and the receivers.)

3.3. Bootstrapping Receivers

Before a receiver can authenticate messages with TESLA, it needs to have:

- An upper bound D_t on the lag of its own clock with respect to the clock of the sender. (That is, if the local time reading is t, the current time reading at the sender is at most t+D_t.).

- One authenticated key of the one-way key chain. (Typically, this will be the last key in the chain, i.e. K_0, this key will be signed by the sender, and all receivers will verify the signature against the public key of the signer.

- The disclosure schedule of keys:

    - T_int, the interval duration.
    - T_0, the start time of interval 1.
    - N, the length of the one-way key chain.
    - d, the key disclosure delay d (in number of intervals).

The receiver can perform the time synchronization and getting the authenticated TESLA parameters in a two-round message exchange, as described below. We stress again that time synchronization can be performed as part of the registration protocol between any receiver (including late joiners) and the sender, or between any receiver and a group controller.

3.3.1. Time Synchronization

Various approaches exist for time synchronization [16,17,18,19]. TESLA only requires the receiver to know an upper bound on the delay of its local clock with respect to the sender's clock, so a simple algorithm is sufficient. TESLA can be used with direct, indirect, and delayed synchronization as three default options. The specific synchronization method will be part of each instantiation of TESLA.

For completeness, we sketch a simple method for direct synchronization between the sender and a receiver:

- The receiver sends a (sync t_r) message to the sender and records its local time t_r at the moment of sending.
- Upon receipt of the (sync t_r) message, the sender records its local time t_s and sends (synch, t_r,t_s) to the receiver.
- Upon receiving (synch,t_r,t_s), the receiver sets D_t = t_s - t_r + S, where S is an estimated bound on the clock drift throughout the duration of the session.

Note:

- Assuming that the messages are authentic (i.e., the message received by the receiver was actually sent by the sender), and assuming that the clock drift is at most S, then at any point throughout the session we have that T_s < T_r + D_t, where T_s is the current time at the sender and T_r is the current time at the receiver.
- The exchange of sync messages needs to be authenticated. This can be done in a number of ways, for instance a secure NTP protocol, or in conjunction with a session set-up protocol.

For indirect time synchronization (eg, synchronization via a group controller), the sender and the controller engage in a protocol for finding the value D^0_t between the sender and the controller. Next, each receiver R interacts with the group controller (say, when registering to the group) and finds the value D^R_t between the group controller and R. The overall value of D_t within R is set to the sum D_t = D^R_t + D^0_t.

3.4. Broadcasting Authenticated Messages

Each key in the one-way key chain corresponds to a time interval. Every time a sender broadcasts a message, it appends a MAC to the message, using the key corresponding to the current time interval. The key remains secret for the next d-1 intervals, so messages a

sender broadcasts in interval j effectively disclose key K_j-d. We call d the key disclosure delay.

We do not want to use the same key multiple times in different cryptographic operations, that is, to use key K_j to derive the previous key of the one-way key chain K_{j-1}, and to use the same key K_j as the key to compute the MACs in time interval j may potentially lead to a cryptographic weakness. Using a pseudo-random function (PRF) f', we construct the one-way function F': F'(k) = f'_k(1). We use F' to derive the key to compute the MAC of messages in each interval. The sender derives the MAC key as follows: K'_i = F'(K_i). Figure 1 depicts the one-way key chain construction and MAC key derivation. To broadcast message M_j in interval i the sender constructs the packet

$$P_j = \{M_j \,||\, i \,||\, MAC(K'_i, M_j) \,||\, K_{i-d}\}$$

where || denotes concatenation.

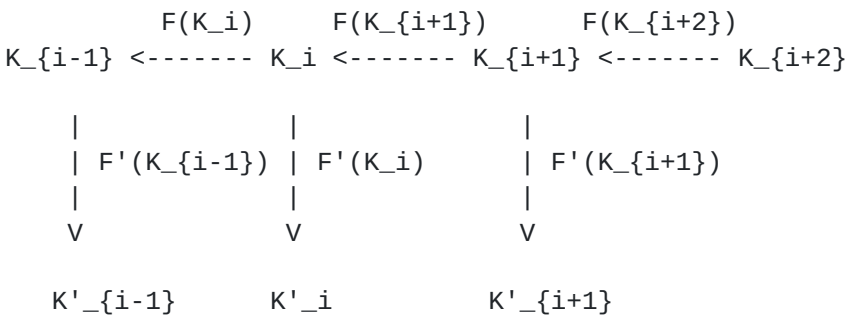

Figure 1: At the top of the figure, we see the one-way key chain (derived using the one-way function F), and the derived MAC keys (derived using the one-way function F').

3.5. Authentication at Receiver

Once a sender discloses a key, we must assume that all parties might have access to that key. An adversary could create a bogus message and forge a MAC using the disclosed key. So whenever a packet arrives, the receiver must verify that the MAC is based on a safe key; a safe key is one that is still secret (only known by the sender). We define a safe packet or safe message to be one with a MAC that is computed with a safe key.

If a packet proves safe it will be buffered, only to be released when its own key, disclosed in a later packet, proves its authenticity. Although a newly arriving packet cannot immediately be authenticated, it may disclose a new key so that earlier, buffered packets can be authenticated. Any newly disclosed key must be checked to determine whether it is genuine, then authentication of buffered packets that have been waiting for it can proceed.

We now describe TESLA authentication at the receiver with more

precision, listing all of these steps in the exact order they should be carried out:

1. Safe packet test: When the receiver receives packet P_j which carries an interval index i, and a disclosed key K_{i-d}, it first records local time T at which the packet arrived. The receiver then computes an upper bound t_j on the sender's clock at the time when the packet arrived: t_j = T + D_t. To test whether the packet is safe, the receiver then computes the highest interval x the sender could possibly be in, namely x = floor((t_j - T_0) / T_int). The receiver verifies that x < i + d (where i is the interval index), which implies that the sender is not yet in the interval during which it discloses the key K_i.

   Even if the packet is safe, the receiver cannot yet verify the authenticity of this packet sent in interval i without key K_i that will be disclosed later. Instead, it adds the triplet (i, M_j, MAC(K'_i, M_j)) to a buffer, and verifies the authenticity after it learns K'_i.

   If the packet is unsafe, then the receiver considers the packet unauthenticated. It should discard unsafe packets but, at its own risk it may choose to use them unverified.

2. New key index test: Next the receiver checks whether a key K_v has already been disclosed with the same or later index v than the current disclosed key K_{i-d}, that is with v >= i-d.

3. Key verification test: If the disclosed key index is new, the receiver checks the legitimacy of K_{i-d} by verifying, for some earlier disclosed key K_v (v<i-d), that K_v = F^{i-d-v}(K_{i-d}).

   If key verification fails, the newly arrived packet P_j should be discarded.

4. Message verification tests: If the disclosed key is legitimate, the receiver then verifies the authenticity of any earlier safe, buffered packets of interval i-d. To authenticate one of the buffered packets P_h containing message M_h protected with a MAC that used key index i-d, the receiver will compute K'_{i-d} = F'(K_{i-d}) from which it can compute MAC(K'_{i-d}, M_h).

   If this MAC equals the MAC stored in the buffer, the packet is authenticated and can be released from the buffer. If the MACs do not agree, the buffered packet P_h should be discarded.

   The receiver continues to verify and release (or not) any remaining buffered packets that depend on the newly disclosed key K_{i-d}.

Using a disclosed key, we can calculate all previous disclosed keys, so even if packets are lost, we will still be able to verify buffered, safe packets from earlier time intervals. Thus, if i-d-v>1, the receiver can also verify the authenticity of the stored packets of intervals v+1 ... i-d-1.

3.6. Determining the Key Disclosure Delay

An important TESLA parameter is the key disclosure delay d. Although the choice of the disclosure delay does not affect the security of the system, it is an important performance factor. A short disclosure delay will cause packets to lose their safety property, so receivers will not be able to authenticate them; but a long disclosure delay leads to a long authentication delay for receivers. We recommend determining the disclosure delay as follows: in direct time synchronization let the RTT, 2m, be a reasonable upper bound on the round trip time between the sender and any receiver including worst case congestion delay and worst case buffering delay in host stacks. Then choose d = ceil(2m / T_int) + 1. Note that rounding up the quotient ensures that d >= 2. Also note that a disclosure delay of one time interval (d=1) does not work. Consider packets sent close to the boundary of the time interval: after the network propagation delay and the receiver time synchronization error, a receiver will not be able to authenticate the packet, because the sender will already be in the next time interval, when it discloses the corresponding key.

Measuring the delay to each receiver before determining m will still not adequately predict the upper bound on delay to late joiners, or where congestion delay rises later in the session. It may be adequate to use a hard-coded historic estimate of worst-case delay (e.g. round trip delays to any host on the intra-planetary Internet rarely exceed 500msec if routing remains stable). If such authentication delay is too pessimistic, the adaptive approach of section 3.7 may be an alternative, at the expense of extra packet overhead.

We stress that the security of TESLA does not rely on any assumptions on network propagation delay: If the delay is longer than expected then authentic packets may be considered unauthenticated. Still, no inauthentic packet will be accepted as authentic.

3.7. An alternative delay description method

The above description instructs the sender to include the time interval i in each packet. The receiver then uses i to determine the time at which the key authenticating the packet is disclosed. This method limits the the sender to a pre-determined schedule of disclosing keys.

Alternatively, the sender may directly include in each packet the time t_p at which it is going to disclose the key for this packet. This way, the receiver does not need to know the duration of intervals or the delay factor d. All the receiver needs to know is the bound D_t on the clock skew and T_0, the sender's local time at the initiation of the session. Then the receiver records the local time T when the packet has arrived, and verifies that

$$T <= T_0 + D_t + t_p.$$

Else the packet is considered unauthenticated.

Another advantage of this method is that the sender is able to change the duration of intervals and the key disclosure delay dynamically throughout the session. It is stressed, however, that the interval index i must still be included in the packet, to allow the receiver to know which key K_i should be used to verify the packet.

3.8. Denial of service protection

Because TESLA authentication is delayed, receivers seem vulnerable to flooding attacks that cause them to buffer excess packets, even though they may eventually prove to be inauthentic. When TESLA is deployed in an environment with a threat of flooding attacks, the receiver can take a number of extra precautions.

First we list simple DoS mitigation precautions that can and should be taken by any receiver independently of others, thus requiring no changes to the protocol or sender behaviour. We precisely specify where these extra steps interleave with the receiver authentication steps already given in section 3.5.

áá o Session validity test: Before the safe packet test (step 1), check that arriving packets have a valid source IP address and port number for the session, that they do not replay a message already received in the session and that they are not significantly larger than the packet sizes expected in the session.

áá o Reasonable misordering test: Before the key verification test (step 3), check the disclosed key index i-d of the arriving packet is within g of the previous highest disclosed key index v, thus for example i-d-v <= g. g sets the threshold beyond which an out of order key index is assumed to be malicious rather than just misordered. Without this test an attacker could exploit the iterated test in step 3 to make receivers consume inordinate CPU time checking along the hash chain for what appear to be extremely misordered packets.

Each receiver can independently adapt g to prevailing attack

conditions, for instance using the following algorithm. Initially,

g should be set to g_max (say 16). But whenever an arriving packet fails the reasonable misordering test above or the key verification test (step 3), g should be dropped to g_min (>0 and typically 1). At each successful key verification (step 3), g should be incremented by 1 unless it is already g_max. These precautions will guarantee that sustained attack packets cannot cause the receiver to execute more than an average of g_min hashes each, unless they are paced against genuine packets. In the latter case attacks are limited to g_max/(g_max-g_min) hashes per each genuine packet.

g_max and g_min should be chosen knowing that they limit the average gap in a packet sequence to g.max(n,m)/n packets (see section 3.2 for definitions of n & m). So with g=1, m=100msec RTT and n=4msec inter-packet period, reordering would be limited to gaps of 25 packet on average. Bigger naturally occurring gaps would have to be written off as if they were losses.

Stronger DoS protection requires both senders and receivers to arrange additional constraints on the protocol. Below we outline three alternative extensions to basic TESLA; the first adding group authentication, the second not re-using keys during a time interval and the third moving buffering to the sender.

It is important to understand the applicability of each scheme, as the first two schemes use slightly more (but bounded) resources in order to prevent attackers from consuming unbounded resources. Adding group authentication requires larger per packet overhead. Never re-using a key requires both ends to process two hashes per packet (rather than per time interval) and the sender must store or re-generate a longer hash chain. The merits of each scheme, summarised after describing each below, must be weighed against these additional costs.

3.8.1. Additional group authentication

This scheme simply involves addition of a group MAC to every packet. That is, a shared key K_g common to the whole group is communicated as an additional step during receiver bootstrap (section 3.3). Then, during broadcast of message M_j (section 3.4) the sender computes the group MAC of each packet MAC(K_g, P_j), which it appends to the packet header. Note that the group MAC covers the whole packet P_j, that is the concatenation of the message M_j and the additional TESLA authentication material, using the formula in section 3.4. Immediately on packet arrival, each receiver can check that each packet came from a group member, by recomputing and comparing the group MAC.

It should be noted that TESLA source authentication is only necessary

when other group members cannot be trusted to refrain from spoofing the source, otherwise simpler group authentication would be

sufficient. Therefore, additional group authentication will only make sense in scenarios where other group members are trusted to refrain from flooding the group, but they are still not trusted to refrain from spoofing the source.

3.8.2. Not re-using keys

In TESLA as described so far, each MAC key was used repeatedly for all the packets sent in a time interval. If instead the sender were to guarantee never to use a MAC key more than once, each disclosed key could assume an additional purpose on top of authenticating a previously buffered packet. Each key would also immediately show each receiver that the sender of each arriving packet knew the next key back along the hash chain, which is now only ever disclosed once, similar to S/KEY [23]. Therefore a reasonable receiver strategy would be to discard any arriving packets that disclosed a key seen already. The fill rate of the receiver's buffer would then be clocked by each packet revealed by the genuine sender, preventing memory flooding attacks.

An attacker with control of a network element or of a faster bypass network could intercept messages and overtake or replace them with different messages but the same keys. However, as long as packets are only buffered if they also pass the delay safety test, such bogus packets will fail TESLA verification after the disclosure delay. Admittedly, receivers could be fooled into discarding genuine messages that had been overtaken by bogus ones. But it is hard to overtake messages without compromising a network element. And any attacker that can compromise a network element can discard genuine messages anyway. We will now describe this scheme in more detail.

For the sender the scheme is hardly different from TESLA. It merely uses an interval duration short enough to ensure a new key back along the hash chain for each packet. So the rule of thumb given in section 3.2 for an efficient re-keying interval T_int no longer applies. Instead, T_int is simply n, the inter-arrival time between packets in milliseconds. The rule of thumb for calculating d, the key disclosure delay, remains unchanged from that given in section 3.6, or the explicit disclosure delay method in section 3.7 can be used.

If the packet rate is likely to vary, for safety n should be taken as the minimum inter-departure time between any two packets. (In fact, n need not be so strict; it can be the minimum average packet inter-departure time over any burst of d packets expected throughout the session.)

Note that if the packet rate slows down, whenever no packets are sent in a key change interval the key index must increment along the hash

chain once for each missed interval. (During a burst, if the less strict definition of n above has been used, packets may need to

depart before their key change interval. The sender can safely continue changing the key each packet, using keys from future key intervals, because if n has been chosen as defined above, such bursts will never sustain long enough to cause the associated key to be disclosed less than the disclosure delay later.)

To be absolutely clear, the precise guarantees that the sender keeps to by following the above guidance are:

- not to re-use a MAC key
- not to use a MAC key K_i after its time interval i
- not to disclose key K_i sooner than the disclosure delay d * T_int following the packet it protects

Sender setup, receiver bootstrapping and broadcasting authenticated messages are otherwise all identical to the descriptions in sections 3.2, 3.3 and 3.4 respectively. However, the following step must be added to the receiver authentication steps in section 3.5:

- After step 2, if a packet arrives carrying a key index i-d that has already been received, it should not be buffered.

This simple scheme would suffice against DoS, were it not for the fact that a network sometimes misorders packets without being compromised. Even without control of a network element, an attacker can opportunistically exploit such openings to fool a receiver into buffering a bogus packet and discarding a later genuine one. A receiver can choose to set aside a fixed size cache and manage it to minimise the chances of discarding a genuine packet. However, given such vulnerabilities are rare and unpredictable, it is simpler to count these events as additions to the network loss rate. As always, TESLA authentication will still uncover any bogus packets after the disclosure delay.

To summarise, avoiding re-using keys has the following properties, even under extreme flooding attacks:

- After delayed TESLA authentication, packets arriving within the disclosure delay will always be identified as authentic if they are and inauthentic if they are not.
- The fill rate of the receiver's buffer is clocked by each packet revealed by the genuine sender, preventing memory flooding attacks.
- An attacker with control of a network element can cause any loss rate it chooses (but that's always true anyway).
- Where attackers do not have control of any network elements, the

effective loss rate is bounded by the sum of the network's actual loss rate and its re-ordering rate.

3.8.3. Sender buffering

Buffering of packets can be moved to the sender side, then receivers can authenticate packets immediately upon receipt. This method is described in [15].

3.9. Some extensions

Let us mention two salient extensions of the basic TESLA scheme. A first extension allows having multiple TESLA authentication chains for a single stream, where each chain uses a different delay for disclosing the keys. This extension is typically used to deal with heterogeneous network delays within a single multicast transmission. A second extension allows having most of the buffering of packets at the sender side (rather than at the receiver side). Both extensions are described in [15].

The requirement in TESLA to receive a key in a later packet for authentication prevents a receiver from authenticating the last part of a message. Thus, to enable authentication of the last part of a message or of the last message before a transmission suspension, the sender needs to send an empty message with the key to enable authentication.

4. Layer placement

TESLA authentication can be performed at any layer in the networking stack. Three natural places are in the network, transport, or the application layer. We list some considerations regarding the choice of layer:

- Performing TESLA in the network layer has the advantage that the transport or application layer only receives authenticated data, potentially aiding a reliability protocol and mitigating denial of service attacks. (Indeed, reliable multicast tools based on forward error correction are highly susceptible to denial of service due to bogus packets.)

- Performing TESLA in either the transport or the application layer has the advantage that the network layer remains unchanged; but it has the potential drawback that packets are obtained by the application layer only after being processed by the transport layer. Consequently, if buffering is used in the transport then this may introduce additional and unpredictable delays on top of the unavoidable network delays.

- It should be kept in mind that, since TESLA relies upon timing of packets, deploying TESLA on top of a protocol or layer which aggressively buffers packets and hides the true packet arrival

time will significantly reduce TESLA's performance.

5. IANA Considerations

This document has no actions for IANA.

6. Security Considerations

See the academic publications on TESLA [8,14,20] for several security analyses. Regarding the security of implementations, by far the most delicate point is the verification of the timing conditions. Care should be taken to make sure that: (a) the value bound D_t on the clock skew is calculated according to the spec at session set-up; (b) the receiver records the arrival time of the packet as soon as possible after the packet's arrival, and computes the safety condition correctly.

Finally, in common with all authentication schemes, if verification is located separately from the ultimate destination application (e.g. an IPSec tunnel end point), a trusted channel must be present between verification and the application. For instance, the interface between the verifier and the application might simply assume that packets received by the application must have been verified by the verifier (because otherwise they would have been dropped). The application is then vulnerable to reception of packets that have managed to bypass the verifier.

7. Acknowledgments

We would like to thank the following for their feedback and support: Mike Luby, Mark Baugher, Mats Naslund, Dave McGrew, Ross Finlayson, Sylvie Laniepce, Lakshminath Dondeti, Russ Housley and the IESG reviewers

8. References

All references are informative.

[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] D. Boneh, G. Durfee, and M. Franklin, "Lower bounds for multicast message authentication," in Advances in Cryptology -- EUROCRYPT '2001 (B. Pfitzmann, ed.), vol. 2045 of Lecture Notes in Computer Science , (Innsbruck, Austria), pp. 434--450, Springer-Verlag, Berlin Germany, 2001.

[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] 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.

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

[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] 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.

[13] A. Perrig and J. D. Tygar, Secure Broadcast Communication in Wired and Wireless Networks Kluwer Academic Publishers, Oct. 2002. ISBN 0792376501.

[14] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, "The tesla broadcast authentication protocol," RSA CryptoBytes , vol. 5, no. Summer, 2002.

[15] A. Perrig, R. Canetti, D. Song, and J. D. Tygar, "Efficient and secure source authentication for multicast," in Network and Distributed System Security Symposium, NDSS '01 , pp. 35--46, February 2001.

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

[17] B. Simons, J. Lundelius-Welch, and N. Lynch, "An overview of clock synchronization," in Fault-Tolerant Distributed Computing (B. Simons and A. Spector, eds.), no. 448 in LNCS, pp. 84--96, Springer-Verlag, Berlin Germany, 1990.

[18] D. Mills, "Improved algorithms for synchronizing computer network clocks," in Proceedings of the conference on Communications architectures, protocols and applications, SIGCOMM 94 , (London, England), pp. 317--327, 1994.

[19] L. Lamport and P. Melliar-Smith, "Synchronizing clocks in the presence of faults," J. ACM , vol. 32, no. 1, pp. 52--78, 1985.

[20] Philippa Broadfoot and Gavin Lowe, "Analysing a Stream Authentication Protocol using Model Checking. In Proceedings of the 7th European Symposium on Research in Computer Security (ESORICS), 2002.

[21] M. Jakobsson, "Fractal hash sequence representation and traversal." Cryptology ePrint Archive, http://eprint.iacr.org/2002/001/, Jan. 2002.

[22] D. Coppersmith and M. Jakobsson, "Almost optimal hash sequence traversal," in Proceedings of the Sixth International Financial Cryptography Conference (FC '02) , March 2002.

[23] N. Haller, "The S/KEY one-time password system," IETF RFC 1760, February 1995.

A. Author Contact Information

Adrian Perrig
ECE Department
Carnegie Mellon University
Pittsburgh, PA 15218
US
perrig@cmu.edu

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

Dawn Song
ECE Department
Carnegie Mellon University
Pittsburgh, PA 15218
US
dawnsong@cmu.edu

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

Bob Briscoe
BT Research
B54/77, BT Labs
Martlesham Heath
Ipswich, IP5 3RE
UK
bob.briscoe@bt.com

B. Full Copyright Statement, IPR Notice and Disclaimer

Change log (to be removed before publication)

Main changes from draft-03 are:

* Abstract

  Completely re-written to fit length guidelines

* Section 2. Functionality:

  Added some clarification of the properties of TESLA

* Section 3. The Basic TESLA Protocol

  Clarified why other TESLA publications are not normative references.

* Section 3.2 Sender Setup

  Removed the duplicate rule of thumb for determining d, and replaced with reference forward to section 3.6. Replaced the imprecise definition of m with half the precise definition of RTT from section 3.6:

  o Section 3.2 said d = ceil(2m/T_int);

  o Section 3.6 said d = ceil(RTT/T_int) + 1

  o Section 3.2 defined m as "the average network delay"

  o Section 3.6 defined RTT as "a reasonable upper bound on the round trip time between the sender and any receiver"

  Added extra discussion of choice of key interval wrt congestion and late joiners.

* Section 3.3 Bootstrapping Receivers

  Clarified that any receiver including late joiners can do time synchronization independently of others.

* Section 3.5 Receiver authentication

  I had to completely re-arrange this to better allow the DoS section to refer back to it. This was tough. Firstly, I enumerated each step and gave them names (which should also help when other standards refer to this one), so I could interleave steps later in the DoS section. Secondly, the original text focused on one packet, staying with it during buffering then eventual

verification. Instead, I traced the procedures that would be

triggered as a packet arrived. This involved leaving it in the buffer for later and tracing through to the earlier packets it released from the buffer.

* Section 3.6 Determining the Key Disclosure Delay

  Added extra discussion of choice of disclosure delay wrt congestion and late joiners.

* Section 3.8 Denial of service protection

  Added whole new section

* Section 6 Security Considerations

  Removed high level discussion of DoS

* Throughout

  Spell-checked, fixed cross-referencing & nits, formatted.

* Fixed all the other IESG comments in

  <https://datatracker.ietf.org/public/pidtracker.cgi?command=print_ballot&ballot_id=690&filename=draft-ietf-msec-tesla-intro>

  IANA considerations, boilerplate stuff, Acknowledgements, other content issues covered in the relevant sections.