The TESLA Broadcast Authentication Protocol*

Adrian Perrig Ran Canetti J. D. Tygar Dawn Song

Abstract

One of the main challenges of securing broadcast communication is *source authentication*, or enabling receivers of broadcast data to verify that the received data really originates from the claimed source and was not modified en route. This problem is complicated by mutually untrusted receivers and unreliable communication environments where the sender does not retransmit lost packets.

This article presents the TESLA (Timed Efficient Stream Loss-tolerant Authentication) broadcast authentication protocol, an efficient protocol with low communication and computation overhead, which scales to large numbers of receivers, and tolerates packet loss. TESLA is based on loose time synchronization between the sender and the receivers.

Despite using purely symmetric cryptographic functions (MAC functions), TESLA achieves asymmetric properties. We discuss a PKI application based purely on TESLA, assuming that all network nodes are loosely time synchronized.

1 Introduction

Broadcast communication is gaining popularity for efficient and large-scale data dissemination. Examples of broadcast distribution networks are satellite broadcasts, wireless radio broadcast, or IP multicast. While many broadcast networks can efficiently distribute data to multiple receivers, they often also allow a malicious user to impersonate the sender and inject broadcast packets — we call this a *packet injection attack*. (Source-Specific Multicast (SSM, EXPRESS) is a notable exception, and attempts to prevent this attack [17, 40].)

Because malicious packet injection is easy in many broadcast networks, the receivers want to ensure that the broadcast packets they receive really originate from the claimed source. A *broadcast authentication protocol* enables the receivers to verify that a received packet was really sent by the claimed sender.

Simply deploying the standard point-to-point authentication mechanism (i.e., appending a message authentication code (MAC) to each packet, computed using a shared secret key) does not provide secure broadcast authentication. The problem is that any receiver with the secret key can forge data and impersonate the sender. Consequently, it is natural to look for solutions based on asymmetric cryptography to prevent this attack; a digital signature scheme is an example of an asymmetric cryptographic protocol. Indeed, signing each data packet provides secure broadcast authentication; however, it has high overhead, both in terms of the time required to sign and verify, and in terms of the bandwidth. Several schemes were proposed that mitigate this overhead by amortizing a single signature over several packets, e.g., [14, 25, 28, 33, 38, 39]. However, none of these schemes is fully satisfactory in terms of bandwidth overhead, processing time, scalability, robustness to denial-of-service attacks, and robustness to packet loss. Even though some schemes

*Most of this work was done at UC Berkeley and IBM Research. The authors can be reached at
`adrian+@cs.cmu.edu`, `canetti@watson.ibm.com`, `tygar@cs.berkeley.edu`, `skyxd@cs.cmu.edu`.

amortize a digital signature over multiple data packets, a serious denial-of-service attack is usually possible where an attacker floods the receiver with bogus packets supposedly containing a signature. Since signature verification is often computationally expensive, the receiver is overwhelmed verifying bogus signatures.

Researchers proposed information-theoretically secure broadcast authentication mechanisms [10, 11, 12, 13, 20, 34, 35, 36]. These protocols have a high overhead in large groups with many receivers.

Canetti et al. construct a broadcast authentication protocol based on k different keys to authenticate every message with k different MAC's [7]. Every receiver knows m keys and can hence verify m MAC's. The keys are distributed in such a way that no coalition of w receivers can forge a packet for a specific receiver. The security of their scheme depends on the assumption that at most a bounded number (which is on the order of k) of receivers collude.

Boneh, Durfee, and Franklin show that one cannot build a compact collusion resistant broadcast authentication protocol without relying on digital signatures or on time synchronization [4]. They show that any secure broadcast authentication protocol with per-packet overhead slightly less than the number of receivers can be converted into a signature scheme.

Another approach to providing broadcast authentication uses only symmetric cryptography, more specifically on message authentication codes (MACs), and is based on delayed disclosure of keys by the sender. This technique was independently discovered by Cheung [8] in the context of authenticating link state routing updates. A related approach was used in the *Guy Fawkes* protocol for interactive unicast communication [1]. In the context of multicast streamed data it was proposed by several authors [2, 3, 5, 27, 28].

The main idea of TESLA is that the sender attaches to each packet a MAC computed with a key k known only to itself. The receiver buffers the received packet without being able to authenticate it. A short while later, the sender discloses k and the receiver is able to authenticate the packet. Consequently, a single MAC per packet suffices to provide broadcast authentication, provided that the receiver has synchronized its clock with the sender ahead of time.

This article is an overview of the TESLA broadcast authentication protocol. A more detailed description is in a forthcoming book [30] and in our earlier publications [27, 28]. A standardization effort for TESLA is under way in the Multicast Security (MSEC) working group of the IETF [26]. TESLA is used in a wide variety of applications, ranging from broadcast authentication in sensor networks [29], to authentication of messages in ad hoc network routing protocols [18].

2 Background and Assumptions

TESLA requires that the receivers are loosely time synchronized with the sender. In this section, we review a simple protocol to achieve this time synchronization. TESLA also needs an efficient mechanism to authenticate keys at the receiver — we first review one-way chains for this purpose.

2.1 One-Way Chains

Many protocols need to commit to a sequence of random values. For this purpose, we repeatedly use a one-way hash function to generate a *one-way chain*. One-way chains are a widely-used cryptographic primitive. One of the first uses of one-way chains was for one-time passwords by Lamport [21]. Haller later used the same approach for the S/KEY one-time password system [16]. One-way chains are also used in many other applications.

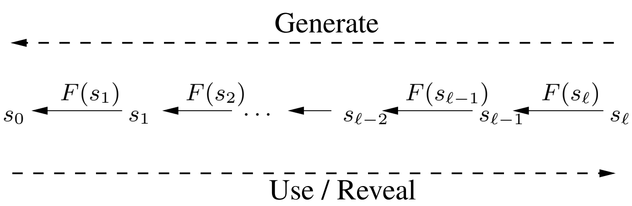

Figure 1: One-way chain example. The sender generates this chain by randomly selecting s_ℓ and repeatedly applying the one-way function F. The sender then reveals the values in the opposite order.

Figure 1 shows the one-way chain construction. To generate a chain of length ℓ we randomly pick the last element of the chain s_ℓ. We generate the chain by repeatedly applying a one-way function F. Finally, s_0 is a commitment to the entire one-way chain, and we can verify any element of the chain through s_0, e.g. to verify that element s_i is indeed the element with index i of the hash chain, we check that $F^i(s_i) = s_0$. More generally, s_i commits to s_j if $i < j$ (to verify that s_j is part of the chain if we know that s_i is the ith element of the chain, we check that $F^{j-i}(s_j) = s_i$). We reveal the elements of the chain in this order $s_0, s_1, \dots, s_{\ell-1}, s_\ell$. How can we store this chain? We can either create it all at once and store each element of the chain, or we can just store s_ℓ and compute any other element on demand. In practice, a hybrid approach helps to reduce storage with a small recomputation penalty. Jakobsson [19], and Coppersmith and Jakobsson [9] propose a storage efficient mechanism for one-way chains: a one-way chain with N elements only requires $\log(N)$ storage and $\log(N)$ computation to access an element.

In TESLA, the elements of the one-way chain are keys, so we call the chain a *one-way key chain*. Furthermore, any key of the one-way key chain commits to all following keys, so we call such a key a *one-way key chain commitment*, or simply *key chain commitment*.

2.2 Time Synchronization

TESLA does not need the strong time synchronization properties that sophisticated time synchronization protocols provide [22, 24, 37], but only requires loose time synchronization, and that the receiver knows an upper bound on the sender's local time. We now outline a simple and secure time synchronization protocol that achieves this requirement. For simplicity, we assume the clock drift of both sender and receiver is negligible (otherwise the receiver can periodically resynchronize the time with the sender). We denote the real difference between the sender and the receiver's time with δ. In loose time synchronization, the receiver does not need to know the exact δ but only an upper bound on it, Δ, which we also refer to as the *maximum time synchronization error*.

We now describe a simple protocol for time synchronization, where each receiver performs explicit time synchronization with the sender. This approach does not require any extra infrastructure to perform time synchronization. We present a simple two-round time synchronization protocol that satisfies the requirement for TESLA, which is that the receiver knows an upper bound on the sender's clock. Reiter previously describes this protocol [31, 32].

Figure 2 shows a sample time synchronization between the receiver and the sender. In the protocol, the receiver first records its local time t_R and sends a time synchronization request containing a nonce to the sender.[1] Upon receiving the time synchronization request, the sender records its local time t_S and replies with a signed response packet containing t_S and the nonce.[2]

[1]The security of this time synchronization protocol relies on the unpredictability of the nonce — if an attacker could predict the receiver's nonce, it could send a time synchronization request to the sender with that nonce, and replay the response later to the receiver.

[2]Interestingly, the processing and propagation delay of the response message does not change δ (assuming that the sender im-

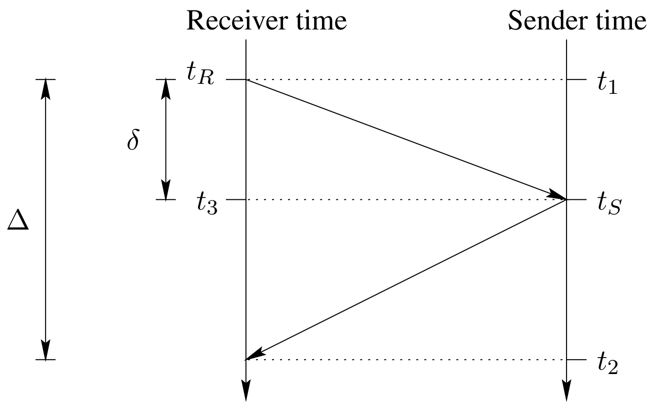

Figure 2: Direct time synchronization between the sender and the receiver. The receiver issues a time synchronization request at time t_R, at which time the sender's clock is at time t_1. The sender responds to the request at its local time t_S. In TESLA, the receiver is only interested in an upper bound on the sender's time. When the receiver has its current time t_r, it computes the upper bound on the current sender's time as $t_s \le t_r - t_R + t_S$. The real synchronization error after this protocol is δ. The receiver, however, does not know the propagation delay of the time synchronization request packet, so it must assume that the time synchronization error is Δ (or the full round-trip time (RTT)).

1. **Setup**. The sender S has a digital signature key pair, with the private key K_S^{-1} and the public key K_S. We assume a mechanism that allows a receiver R to learn the authenticated public key K_S. The receiver chooses a random and unpredictable nonce.

2. **Protocol steps**. Before sending the first message, the receiver records its local time t_R.

   $$R \rightarrow S : \mathsf{Nonce}$$
   $$S \rightarrow R : \{\mathsf{Sender\ time}\ t_S, \mathsf{Nonce}\}_{K_S^{-1}}$$

   To verify the return message, the receiver verifies the digital signature and checks that the nonce in the packet equals the nonce it randomly generated. If the message is authentic, the receiver stores t_R and t_S. To compute the upper bound on the sender's clock at local time t, the receiver computes $t - t_R + t_S$.

Upon receiving the signed response, the receiver checks the validity of the signature and verifies that the nonce in the response packet equals the nonce in the request packet. If all verifications are successful, the receiver uses t_R and t_S to compute the upper bound of the sender's time: when the receiver has the current time t_r, it computes the upper bound on the current sender's time as $t_s \le t_r - t_R + t_S$. The real synchronization error after this protocol is δ, as Figure 2 shows. The receiver, however, does not know the propagation delay of the time synchronization request packet, so it must assume that the time synchronization error is Δ (or the full round-trip time (RTT)).

A digital signature operation is computationally expensive, and we need to be careful about denial-of-service attacks in which an attacker floods the sender with time synchronization requests. Another problem is *request implosion*: the sender is overwhelmed with time synchronization requests from receivers. We address these issues in our earlier paper [27].

3 The TESLA Broadcast Authentication Protocol

A viable broadcast authentication protocol has the following requirements:

mediately records and replies with the arrival time of the request packet), since the receiver is only interested in an upper bound on the sender's clock. If the receiver were interested in the lower bound on the sender's clock, the processing delay and delay of the response message would matter. For more details on this refer to the more detailed time synchronization description [30].

- 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.
- Robustness to packet loss.
- Scales to a large number of receivers.

The TESLA protocol meets all these requirements with low cost — and it has the following special requirements:

- The sender and the receivers must be at least loosely time-synchronized as outlined in Section 2.2.
- Either the receiver or the sender must buffer some messages.

Despite the buffering, TESLA has a low authentication delay. In typical configurations, the authentication delay is on the order of one round-trip delay between the sender and receiver.

3.1 Sketch of TESLA protocol

We first outline the main ideas behind TESLA. Broadcast authentication requires a source of asymmetry, such that the receivers can only verify the authentication information, but not generate valid authentication information. TESLA uses time for asymmetry. We assume that receivers are all loosely time synchronized with the sender — up to some time synchronization error Δ, all parties agree on the current time. Here is a sketch of the basic approach:

- The sender splits up the time into time intervals of uniform duration. Next, the sender forms a one-way chain of self-authenticating values, and assigns the values sequentially to the time intervals (one key per time interval). The one-way chain is used in the reverse order of generation, so any value of a time interval can be used to derive values of previous time intervals. The sender defines a disclosure time for one-way chain values, usually on the order of a few time intervals. The sender publishes the value after the disclosure time.
- The sender attaches a MAC to each packet. The MAC is computed over the contents of the packet. For each packet, the sender determines the time interval and uses the corresponding value from the one-way chain as a cryptographic key to compute the MAC. Along with the packet, the sender also sends the most recent one-way chain value that it can disclose.
- Each receiver that receives the packet performs the following operation. It knows the schedule for disclosing keys and, since the clocks are loosely synchronized, can check that the key used to compute the MAC is still secret by determining that the sender could not have yet reached the time interval for disclosing it. If the MAC key is still secret, then the receiver buffers the packet.
- Each receiver also checks that the disclosed key is correct (using self-authentication and previously released keys) and then checks the correctness of the MAC of buffered packets that were sent in the time interval of the disclosed key. If the MAC is correct, the receiver accepts the packet.

One-way chains have the property that if intermediate values of the one-way chain are lost, they can be recomputed using later values. So, even if some disclosed keys are lost, a receiver can recover the key chain and check the correctness of packets.

The sender distributes a stream of messages $\{M_i\}$, and the sender sends each message M_i in a network packet P_i along with authentication information. The broadcast channel may be lossy, but the sender does not retransmit lost packets. Despite packet loss, each receiver needs to authenticate all the messages it receives.

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

TESLA uses self-authenticating one-way chains. The sender divides the time into uniform intervals of duration T_{int}. Time interval 0 will start at time T_0, time interval 1 at time $T_1 = T_0 + T_{int}$, etc. The sender assigns one key from the one-way chain to each time interval in sequence. The one-way chain is used in the reverse order of generation, so any value of a time interval can be used to derive values of previous time intervals.

The sender determines the length N of the one-way chain $K_0, K_1, \ldots, K_N$, and this length limits the maximum transmission duration before a new one-way chain must be created.[3] The sender picks a random value for K_N. Using a pseudo-random function f, the sender constructs the one-way function F: $F(k) = f_k(0)$. The remainder 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 we can compute any value in the key chain from K_N even if we do not have intermediate values. Each key K_i will be active in time interval i.

[3] For details on how to handle broadcast streams of unbounded duration by switching one-way key chains, see [27]. For this article we assume that chains are sufficiently long for the duration of communication.

3.3 Bootstrapping Receivers

Before a receiver can authenticate messages with TESLA, it needs to be loosely time synchronized with the sender, know the disclosure schedule of keys, and receive an authenticated key of the one-way key chain.

Various approaches exist for time synchronization [24, 37, 22]. TESLA, however, only requires loose time synchronization between the sender and the receivers, so a simple algorithm is sufficient. The time synchronization property that TESLA requires is that each receiver can place an upper bound of the sender's local time, as we discuss in Section 2.2.

The sender sends the key disclosure schedule by transmitting the following information to the receivers over an authenticated channel (either via a digitally signed broadcast message, or over unicast with each receiver):

- Time interval schedule: interval duration T_{int}, start time T_i and index of interval i, length of one-way key chain.
- Key disclosure delay d (number of intervals).
- A key commitment to the key chain K_i ($i = j - d$ where j is the current interval index).

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 sent in interval j effectively disclose key K_{j-d}. We call d the *key disclosure delay*.

As a general rule, using the same key multiple times in different cryptographic operations is ill-advised —

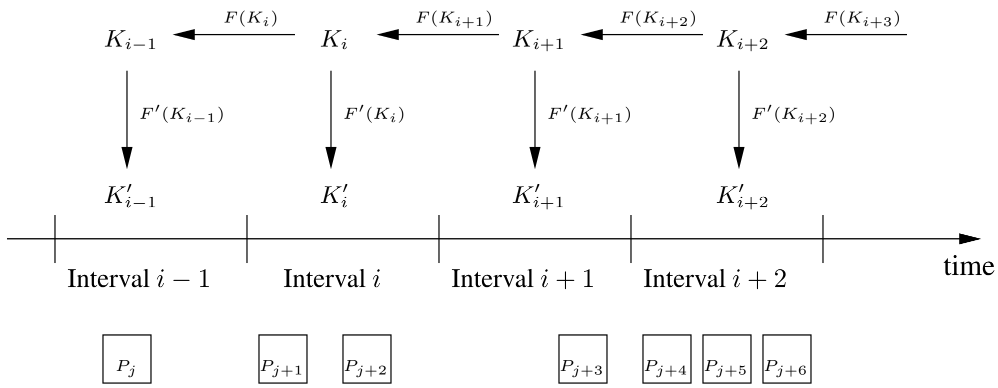

Figure 3: At the top of the figure is the one-way key chain (using the one-way function F), and the derived MAC keys (using the one-way function F'). Time advances left-to-right, and the time is split into time intervals of uniform duration. At the bottom of the figure, we can see the packets that the sender sends in each time interval. For each packet, the sender uses the key that corresponds to the time interval to compute the MAC of the packet. For example for packet P_{j+3}, the sender computes a MAC of the data using key K'_{i+1}. Assuming a key disclosure delay of two time intervals ($d = 2$), packet P_{j+3} would also carry key K_{i-1}.

it may lead to cryptographic weaknesses. So we do not want to use key K_j both to derive key K_{j-1} and to compute MACs. Using a pseudo-random function family 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: $K'_i = F'(K_i)$. Figure 3 depicts the one-way key chain construction and MAC key derivation. To broadcast message M_j in interval i the sender constructs packet $P_j = \{M_j \,||\, \text{MAC}(K'_i, M_j) \,||\, K_{i-d}\}$.

Figure 3 depicts the one-way key chain derivation, the MAC key derivation, the time intervals, and some sample packets that the sender broadcasts.

3.5 Authentication at Receiver

When a sender discloses a key, all parties potentially have access to that key. An adversary can create a bogus message and forge a MAC using the disclosed key. So as packets arrive, the receiver must verify that their MACs are based on *safe keys*: a *safe key* is one that is only known by the sender, and *safe packets* or *safe messages* have MACs computed with safe keys.

Receivers must discard any packet that is not safe, because it may have been forged.

We now explain TESLA authentication in detail: A sender sends packet P_j in interval i. When the receiver receives packet P_j, the receiver can use the self-authenticating key K_{i-d} disclosed in P_j to determine i. It then checks the latest possible time interval x the sender could currently be in (based on the loosely synchronized clock). If $x < i + d$ (recall that d is the key disclosure delay, or number of intervals that the key disclosure is delayed), then the packet is safe. The sender has thus not yet reached the interval where it discloses key K_i, the key that will verify packet P_j.

The receiver cannot yet verify the authenticity of packet P_j sent in interval i. Instead, it adds the triplet $(i, M_j, \text{MAC}(K'_i, M_j))$ to a buffer, and verifies the authenticity after it learns K'_i.

What does a receiver do when it receives the disclosed key K_i? First, it checks whether it already knows K_i or a later key K_j $(j > i)$. If K_i is the latest key received to date, the receiver checks the legitimacy of K_i by verifying, for some earlier key K_v $(v < i)$ that $K_v = F^{i-v}(K_i)$. The receiver then computes $K'_i = F'(K_i)$ and verifies the authenticity of packets of interval i, and of previous intervals if the receiver did not yet receive the keys for these intervals (the receiver can derive them from K_i).

Note that the security of TESLA does not rely on any assumptions on network propagation delay, since each receiver locally determines the packet safety, i.e. whether the sender disclosed the corresponding key. However, if the key disclosure delay is not much longer than the network propagation delay, the receivers will find that the packets are not safe.

4 Discussion

4.1 TESLA Security Considerations

The security of TESLA relies on the following assumptions:

- The receiver's clock is time synchronized up to a maximum error of Δ. (We suggest that because of clock drift, the receiver periodically resynchronizes its clock with the sender.)
- The functions F, F' are secure PRFs, and the function F furthermore provides weak collision resistance.[4]

As long as these assumptions are satisfied, it is computationally intractable for an attacker to forge a TESLA packet that the receivers will authenticate successfully.

[4]See our earlier paper for a formal security proof [28].

4.2 Achieving Asymmetric Security Properties with TESLA

Broadcast authentication requires an asymmetric primitive, which TESLA provides through loosely synchronized clocks and delayed key disclosure. TESLA shares many common properties with asymmetric cryptographic mechanisms. In fact, assuming that all nodes in a network are time synchronized, any key of the key chain serves as a key chain commitment and is similar to a public key of a digital signature: any loosely time synchronized receiver with an authentic key chain commitment can authenticate messages, but not forge a message with a MAC that receivers would accept.

We can construct an efficient PKI based solely on TESLA. Consider an environment with n communicating nodes. We assume that all nodes are loosely time synchronized, such that the maximum clock offset between any two nodes is Δ; and that all nodes know the authentic key chain commitment and key disclosure schedule of the certification authority (CA). We further assume that the CA knows the authentic key chain commitment and key disclosure schedule of every node. If a node A wants to start authenticating packets originating from another node B, A can contact the CA for B's key chain commitment and key disclosure schedule, which the CA sends authenticated with its TESLA instance. After the CA discloses the corresponding key, A can authenticate B's TESLA parameters and subsequently authenticate B's packets.

Note that TESLA is not a signature mechanism and does not provide non-repudiation, as anybody could forge "authentic" TESLA packets after the key is disclosed. However, in conjunction with a trusted time stamping mechanism, TESLA could achieve properties similar to a digital signature. Consider this setup: all nodes in the network are loosely time synchronized (as above with an upper bound on the synchronization error); and all nodes in the network trust the time stamping server [6, 15, 23]. The time stamping server timestamps all TESLA packets it receives. The time stamping server can broadcast the hooks to the trust chain authenticated with its TESLA instance. A judge who wants to verify that a sender sent packet P performs the following operations:

1. Receive the current value of the time stamping server's trust chain, ensure that it is safe, and wait for the TESLA key to authenticate it.
2. Based on the trust chain value, verify that packet P is part of the trust chain.
3. Verify that packet P was safe when the time stamping server received it (not necessary if the time stamping server only timestamps safe packets).
4. Retrieve key from the sender and verify it using the key chain commitment and disclosure schedule recorded by the time stamping server.
5. Verify that the authenticity of the packet, which implies that the correct sender must have generated the packet.

TESLA and a time stamping server can thus achieve non-repudiation. This example also shows that the TESLA authentication can also be performed after the key is already disclosed, as long as the verifier can check that the packet arrived safely.

5 Acknowledgments

We gratefully acknowledge funding support for this research. This research was sponsored in part the United States Postal Service (contract USPS 102592-01-Z-0236), by the United States Defense Advanced Research Projects Agency (contract N66001-99-2-8913), and by the United States National Science Foundation (grants 99-79852 and 01-22599). DARPA Contract N66001-99-2-8913 is under the supervision of the Space and Naval Warfare Systems Center, San Diego.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing official policies, either expressed or implied, of the United States government, of DARPA, NSF, USPS, any of its agencies.

References

[1] R. Anderson, F. Bergadano, B. Crispo, J. Lee, C. Manifavas, and R. Needham. A new family of authentication protocols. *ACM Operating Systems Review*, 32(4):9–20, October 1998.

[2] F. Bergadano, D. Cavagnino, and B. Crispo. Chained stream authentication. In *Selected Areas in Cryptography, 7th Annual International Workshop, SAC 2000*, volume 2012 of *Lecture Notes in Computer Science*, pages 144–157, August 2000.

[3] F. Bergadano, D. Cavalino, and B. Crispo. Individual single source authentication on the mbone. In *ICME 2000*, Aug 2000.

[4] D. Boneh, G. Durfee, and M. Franklin. Lower bounds for multicast message authentication. In *Advances in Cryptology — EUROCRYPT '2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 434–450, 2001.

[5] B. Briscoe. FLAMeS: Fast, Loss-Tolerant Authentication of Multicast Streams. Technical report, BT Research, 2000. `http://www.labs.bt.com/people/briscorj/papers.html`.

[6] A. Buldas, P. Laud, H. Lipmaa, and J. Villemson. Time-stamping with binary linking schemes. In

Advances in Cryptology — CRYPTO '98, volume 1462 of *Lecture Notes in Computer Science*, pages 486–501, 1998.

[7] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast security: A taxonomy and some efficient constructions. In *INFOCOMM'99*, pages 708–716, March 1999.

[8] S. Cheung. An efficient message authentication scheme for link state routing. In *13th Annual Computer Security Applications Conference*, pages 90–98, 1997.

[9] D. Coppersmith and M. Jakobsson. Almost optimal hash sequence traversal. In *Proceedings of the Fourth Conference on Financial Cryptography (FC '02)*, Lecture Notes in Computer Science, 2002.

[10] Y. Desmedt and Y. Frankel. Shared generation of authenticators and signatures. In *Advances in Cryptology — CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 457–469, 1992.

[11] Y. Desmedt, Y. Frankel, and M. Yung. Multi-receiver / multi-sender network security: Efficient authenticated multicast / feedback. In *Proceedings IEEE Infocom '92*, pages 2045–2054, 1992.

[12] Y. Desmedt and M. Yung. Arbitrated unconditionally secure authentication can be unconditionally protected against arbiter's attacks. In *Advances in Cryptology — CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 177–188, 1991.

[13] F. Fujii, W. Kachen, and K. Kurosawa. Combinatorial bounds and design of broadcast authentication. *IEICE Transactions*, E79-A(4):502–506, 1996.

[14] R. Gennaro and P. Rohatgi. How to sign digital streams. In *Advances in Cryptology — CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 180–197, 1997.

[15] S. Haber and W. Stornetta. How to time-stamp a digital document. In *Advances in Cryptology — CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 437–455, 1991.

[16] N. Haller. The S/Key one-time password system. In *Proceedings of the Symposium on Network and Distributed Systems Security*, pages 151–157. Internet Society, February 1994.

[17] H. Holbrook and D. Cheriton. IP multicast channels: EXPRESS support for large-scale single-source applications. In *Proceedings of ACM SIGCOMM '99*, September 1999.

[18] Y.-C. Hu, A. Perrig, and D. B. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. In *Proceedings of the Eighth ACM International Conference on Mobile Computing and Networking (Mobicom 2002)*, September 2002. To appear.

[19] M. Jakobsson. Fractal hash sequence representation and traversal. In *Proceedings of the 2002 IEEE International Symposium on Information Theory (ISIT '02)*, pages 437–444, July 2002.

[20] K. Kurosawa and S. Obana. Characterization of (k,n) multi-receiver authentication. In *Proceedings of the 2nd Australasian Conference on Information Security and Privacy (ACISP '97)*, volume 1270 of *Lecture Notes in Computer Science*, pages 205–215, 1997.

[21] L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, November 1981.

[22] L. Lamport and P. Melliar-Smith. Synchronizing clocks in the presence of faults. *Journal of the ACM*, 32(1):52–78, 1985.

[23] H. Lipmaa. *Secure and Efficient Time-Stamping Systems*. PhD thesis, Department of Mathematics, University of Tartu, Estonia, April 1999.

[24] D. Mills. Network Time Protocol (version 3) specification, implementation and analysis. Internet Request for Comment RFC 1305, Internet Engineering Task Force, March 1992.

[25] S. Miner and J. Staddon. Graph-based authentication of digital streams. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 232–246, May 2001.

[26] Multicast security ietf working group (msec). `http://www.ietf.org/html.charters/msec-charter.html`, 2002.

[27] A. Perrig, R. Canetti, D. Song, and J. D. Tygar. Efficient and secure source authentication for multicast. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS 2001)*, pages 35–46. Internet Society, February 2001.

[28] A. Perrig, R. Canetti, J. D. Tygar, and D. Song. Efficient authentication and signature of multicast streams over lossy channels. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 56–73, May 2000.

[29] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. SPINS: Security protocols for sensor networks. In *Proceedings of Seventh Annual International Conference on Mobile Computing and Networks (Mobicom 2001)*, pages 189–199, 2001.

[30] A. Perrig and J. D. Tygar. *Security Protocols for Broadcast Networks*. Kluwer Academic Publishers, 2002. To appear.

[31] M. Reiter. *A security architecture for fault-tolerant systems*. PhD thesis, Department of Computer Science, Cornell University, August 1993.

[32] M. Reiter, K. Birman, and R. van Renesse. A security architecture for fault-tolerant systems. *ACM Transactions on Computer Systems*, 12(4):340–371, November 1994.

[33] P. Rohatgi. A compact and fast hybrid signature scheme for multicast packet. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, pages 93–100, November 1999.

[34] R. Safavi-Naini and H. Wang. New results on multireceiver authentication codes. In *Advances in Cryptology — EUROCRYPT '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 527–541, 1998.

[35] R. Safavi-Naini and H. Wang. Multireceiver authentication codes: Models, bounds, constructions and extensions. *Information and Computation*, 151(1/2):148–172, 1999.

[36] G. Simmons. A cartesian product construction for unconditionally secure authentication codes that permit arbitration. *Journal of Cryptology*, 2(2):77–104, 1990.

[37] B. Simons, J. Lundelius-Welch, and N. Lynch. An overview of clock synchronization. In B. Simons and A. Spector, editors, *Fault-Tolerant Distributed Computing*, number 448 in LNCS, pages 84–96, 1990.

[38] D. Song, D. Zuckerman, and J. D. Tygar. Expander graphs for digital stream authentication and robust overlay networks. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 258–270, May 2002.

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

[40] Source-Specific Multicast IETF working group (SSM). `http://www.ietf.org/html.charters/ssm-charter.html`, 2002.