Seven Cardinal Properties of Sensor Network Broadcast Authentication*

Mark Luk Adrian Perrig Bram Whillock
Carnegie Mellon University

ABSTRACT

We investigate the design space of sensor network broadcast authentication. We show that prior approaches can be organized based on a taxonomy of seven fundamental proprieties, such that each approach can satisfy at most six of the seven proprieties. An empirical study of the design space reveals possibilities of new approaches, which we present in the following two new authentication protocols: RPT and LEA. Based on this taxonomy, we offer guidance in selecting the most appropriate protocol based on an application's desired proprieties. Finally, we pose the open challenge for the research community to devise a protocol simultaneously providing all seven properties.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General– *Security and protection*; K.6.5 [**Security and Protection**]: Security and Protection – *Authentication*

General Terms

Security, Design

Keywords

Broadcast Authentication, Taxnonomy, Sensor Network

1. INTRODUCTION

Due to the nature of wireless communication in sensor networks, attackers can easily inject malicious data messages or alter the content of legitimate messages during multihop forwarding. Sensor network applications thus need to rely on authentication mechanisms to ensure that data from a valid source was not altered in transit. Authentication is thus arguably the most important security primitive in sensor network communication. *Source authentication* ensures a receiver that the message originates from the claimed sender, and *data authentication* ensures that the data from that sender was unchanged (thus also providing *message integrity*). When we use the term *authentication* we mean both source and data authentication.

Authentication of broadcast data is a challenging problem. Furthermore, it is of central importance as broadcasts are used in many applications. For example, routing tree construction, network query, software updates, time synchronization, and network management functions all rely on broadcast. Without an efficient broadcast authentication algorithm, the base station would have to resort to per-node unicast messages, which does not scale to large networks. The practicality of many secure sensor network applications thus hinges on the presence of an efficient algorithm for broadcast authentication.

In point-to-point authentication, authentication can be achieved through purely symmetric means: the sender and receiver would share a secret key used to compute a cryptographic message authentication code (MAC) over each message [15, 23]. When a message with a valid MAC is received, the receiver can be assured that the message originated from the sender. Researchers showed that MACs can be efficiently implemented on resource-constrained sensor network nodes [31], and find that computing a MAC function requires on the order of 1ms on the computation-constrained Berkeley mote platform [11, 14].

Authentication of broadcast messages in sensor networks is much harder than point-to-point authentication [1]. The symmetric approach used in point-to-point authentication is not secure in broadcast settings, where receivers are mutually untrusted. If all nodes share one secret key, any compromised receiver can forge messages from the sender.

In fact, authenticated broadcast requires an asymmetric mechanism [1]. The traditional approach for asymmetric mechanisms is to use digital signatures, for example the RSA signature [34]. Unfortunately, asymmetric cryptographic mechanisms have high computation, communication, and storage overhead, making their usage on resource-constrained devices impractical for many applications.

The property we need is asymmetry, and many approaches had been suggested for sensor network broadcast authentication. However, objectively comparing such approaches and selecting the most appropriate one for a given application is a non-trivial process, especially for an engineer not specialized in security. The goal of this work is to provide guidance for sensor network broadcast authentication by presenting a systematic investigation of the design space. We arrive at a taxonomy of seven fundamental properties, and present protocols that satisfy all but one property. The list of

*This research was supported in part by CyLab at Carnegie Mellon under grant DAAD19-02-1-0389 from the Army Research Office, and grant CNS-0347807 from the National Science Foundation, and by a gift from Bosch. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of ARO, Bosch, Carnegie Mellon University, NSF, or the U.S. Government or any of its agencies.

SASN'06, October 30, 2006, Alexandria, Virginia, USA.

the desired properties is:

1. Resistance against node compromise,
2. Low computation overhead,
3. Low communication overhead,
4. Robustness to packet loss,
5. Immediate authentication,
6. Messages sent at irregular times,
7. High message entropy.

If we remove *any one* of the above requirements, a viable protocol exists. Table 1 gives an overview of the seven approaches for addressing each case. We show that existing protocols, or small modifications thereof, make up for five of the seven possible cases. We also introduce novel approaches for addressing the final two cases: the RPT protocol to authenticate messages sent at regular times, and the LEA protocol to authenticate low-entropy messages. Finally, we pose the open challenge to the research community to design a broadcast authentication mechanism that satisfies all seven properties.

Outline. The paper is organized as follows. We introduce the taxonomy of seven properties and discuss how current approaches can be organized based on our taxonomy in Section 2. Section 3 describes the μTESLA broadcast authentication protocol and presents several extensions to increase its efficiency and robustness to DoS attacks. In Section 3.3, we introduce RPT, a novel protocol that authenticates synchronous messages. In Section 4, we introduce LEA, a novel protocol for efficient network broadcast authentication for low-entropy messages. Implementation and evaluation is discussed in Section 5. Finally, we present related work in Section 6 and our conclusions and future work in Section 7.

2. TAXONOMY OF EXISTING PROTOCOLS

In this section, we discuss the seven properties of broadcast authentication and describe possible approaches if we were to leave out one of the seven requirements.

Node Compromise. Since sensor nodes are not equipped with tamper-proof or tamper-resistant hardware, any physical attacker would be able to physically compromise a node and obtain its cryptographic keys [5]. Since it is unlikely that tamper-proof hardware will be deployed on sensor motes in the near future, secure sensor network protocols need to be resilient against compromised nodes. However, if the nodes are deployed in a physically secured area (such as an attended army base), or if the application itself is resilient against malicious nodes, node compromise might not be an issue.

If we assume no compromised nodes, all parties could maintain a network-wide key that is used to generate and verify a single Message Authentication Code (MAC) per message. If instead one can assume a low number of compromised nodes, a simple approach exists which uses a different key for each receiver and adds one MAC per receiver to each message. Unfortunately, this approach does not scale to large networks since a 10-byte MACs per receiver would result in prohibitively large messages. To trade off communication overhead with security, researchers propose a multi-MAC approach [3]. In their scheme, the sender chooses some number of random MAC keys, and distributes a subset of keys to each node. Every message carries one MAC with each key (assuming 10 bytes per MAC),[1] which adds a substantial overhead. If an attacker compromises a node, it can only forge a subset of MACs, thus with high probability, other nodes will be able to detect the forgery with their subset of keys. A variant of this approach was used to prevent malicious injection of messages in sensor networks [36, 37].

Computation Overhead. Sensor nodes have limited computation resources, so an ideal protocol would have low computation overhead for both sender and receiver. However, there exist scenarios where computation might not be a particularly critical issue. For example, it is conceivable that certain applications would only require authenticated broadcasts for a small number of packets. In such a case, the application engineer might be willing to allow for a small number of intensive computations.

If we admit a high computation overhead, we can use digital signatures. RSA today requires at least a 1024-bit modulus to achieve a reasonable level of security, and a 2048-bit modulus for a high level of security [18]. ECC can offer the same level of security using 160-bit keys and 224-bit keys, respectively. Recent advancement in ECC signature schemes on embedded processors can perform signature verification using 160-bit ECC keys in about 1 second [10]. Although this represents a dramatic improvement over earlier public key cryptographic schemes [2, 4, 21], signature verification is still 3 orders of magnitude slower than MAC verification, while signature generation is 4 orders of magnitude slower. While we expect future sensor nodes to have more powerful processors, the energy constraints dictated by the limited battery resources will always favor the use of more efficient symmetric cryptographic primitives.

Communication Overhead. Energy is an extremely scarce resource on sensor nodes, and as a result, heavily influences the design of sensor network protocols. In particular, radio communication consumes the most amount of energy, and thus protocols with high communication overhead are avoided if possible. However, in some settings (e.g., powered nodes) energy consumption is not an issue. Thus an authentication protocol that requires high communication overhead would be acceptable.

If we admit a high communication overhead, we can leverage efficient one-time signature constructions that are fast to compute on sensor nodes, but require on the order of 100–200 bytes per signature. Examples include the Merkle-Winternitz (MW) signature which requires 230 bytes per signature [25, 26, 35] (we describe the MW signature in detail in Section 4.1), or the HORS signature, which requires around 100 bytes per signature [33]. The MW signature requires around 200 one-way function computations to verify a signature (which corresponds to roughly 200 ms computation time on a sensor node), while the HORS signature only requires 11 one-way function computations. The disadvantage of the HORS signature is that the public key is about 10 Kbytes,[2] whereas the public key for the MW signature is only 10 bytes. Signature generation is very efficient for both mechanisms, and can be reduced to a single hash function computation assuming a lookup table for the cryptographic values. We leverage the MW signature to construct the LEA broadcast authentication mechanism, which we present in Section 4.

Message Reliability. Our fourth property is message reliability. Reliable message delivery is the property of a network such that valid messages are not dropped. Ultimately, message reliability is an applications issue - some applications require message reliabil-

[1] An 80-bit MAC value achieves security comparable to a 1024-bit RSA signature [18].

[2] This is prohibitively large, since each public key of a one-time signature can be used to authenticate only a single message.

Desired property	Approach if property is relaxed
Resistance to node compromise	Network-wide key
Low computation overhead	Digital signatures
Low communication overhead	One-time signatures
Robustness to packet loss	HORS + chaining of public keys
Immediate authentication	μTESLA
Messages sent at irregular times	RPT, described in Section 3.3
High message entropy	LEA, described in Section 4.2

Table 1: Overview of desired properties of broadcast authentication and approaches. The left column presents the desired property, and the right column presents the approach that achieves all properties but relaxes the property in its left column. The text describes each approach in more detail.

ity, while others do not.

If we have perfect message reliability, we can achieve efficient and immediate authentication by using the HORS signature in a special construction that combines multiple public keys [28]. In this construction, a public key is still 10 Kbytes, but a single public key can be used to authenticate almost arbitrarily many messages, as the public values are incrementally updated as signed messages are sent. The communication and computation costs are the same as for the HORS signature: 1 ms for signature generation, 11 ms for signature verification, and 100 bytes for the signature. Note that in such a scheme, an attacker can start forging HORS signatures if many packets are dropped.

Authentication Delay. Depending on the application, authentication delay may influence the design of the sensor network protocol. For time-critical messages such as fire alarms, the receiver would most likely need to authenticate the message immediately. However, authentication delay is typically acceptable for non-time-critical messages.

If we admit an authentication delay and assume that the receivers are loosely time synchronized with the sender, the μTESLA broadcast authentication protocol only adds a 10 byte MAC and an optional 10 byte key to each message [31]. We review the μTESLA protocol in detail in Section 3.1. To achieve a low computation overhead in the case of infrequent messages sent at unpredictable times, we need to extend the μTESLA protocol to enable fast authentication of the keys in the one-way key chain. In Section 3.2 we present a more efficient key chain construction that enables efficient authentication in this case. Simultaneously, our approach protects μTESLA against denial-of-service attacks by sending bogus key chain values.

Synchronous Messages. Some applications send synchronous messages at regular and predictable times. For example, a key revocation list might be sent to the entire network everyday at noon.

We extend the μTESLA protocol to provide efficient and immediate authentication for synchronous messages sent at regular and predictable times. We name the protocol RPT (Regular-Predictable Tesla), and we present its details in Section 3.3.

Message Entropy. So far, all schemes we describe authenticate unpredictable messages with high entropy. However, in practice, many protocols might only communicate with low-entropy messages. For example, in many applications, there are only a handful of valid commands that a base station can send to a sensor node. Therefore, these command packets could be considered as low-entropy messages.

If we can assure a low upper bound on message entropy, we can leverage one-time signatures in constructions that provide message recovery, where the message is not hashed but directly encoded in the signature. We describe our new LEA protocol in Section 4.

For messages with merely a single bit of entropy, we could employ the following optimization using two hash chains. One hash chain would correspond to messages of '1', while another would correspond to messages of '0'. The sender first sends the last value of both chains to the receivers in an authenticated manner (e.g., using one-time signatures or digital signatures). Next, whenever the sender wishes to send a '0', it would reveal the next value in the hash chain corresponding to '0'. The same is done for the hash chain corresponding to '1'. The receiver needs to keep state of the most recent value it received for each hash chain. Consequently, the receiver can easily verify the authenticity of new values by hashing them and comparing them against the most recent value of each hash chain.

3. BROADCAST AUTHENTICATION WITH THE μTESLA PROTOCOL

In this section, we first present a brief overview of the μTESLA protocol [29], the recommended broadcast authentication protocol if immediate authentication is not required. We improve the μTESLA broadcast authentication protocol to provide efficient authentication for infrequent messages sent at unpredictable times (Section 3.2). In Section 3.3, we describe RPT, further modification of μTESLA that provides immediate authentication for synchronous messages sent at regular and predictable times.

3.1 μTESLA Overview

The TESLA protocol provides efficient broadcast authentication over the Internet which can scale to millions of users, tolerate packet loss, and support real time applications [30]. Currently, TESLA is in the process of being standardized in the MSEC working group of the IETF for multicast authentication.

TESLA has been adapted for broadcast authentication in sensor networks, the resulting protocol is called the μTESLA broadcast authentication protocol [30, 31]. μTESLA is used to secure routing information [17], data aggregation messages [12, 32], etc.

We now overview the μTESLA protocol, a detailed description is available in our earlier paper [31]. Broadcast authentication requires a source of asymmetry, such that the receivers can only verify the authentication information, but not generate valid authentication information. μTESLA uses time for asymmetry. μTESLA assumes that receivers are all loosely time synchronized with the sender – up to some time synchronization error Δ, all parties agree on the current time. Recent research in sensor network time synchronization protocols has made significant progress, resulting in time synchronization accuracy in the range of μs [6, 7], which is much more accurate than the loose time synchronization required by μTESLA. By using only symmetric cryptographic primitives, μTESLA is very efficient and provides practical solutions for resource-constrained sensor networks. Figure 1 shows an example of μTESLA

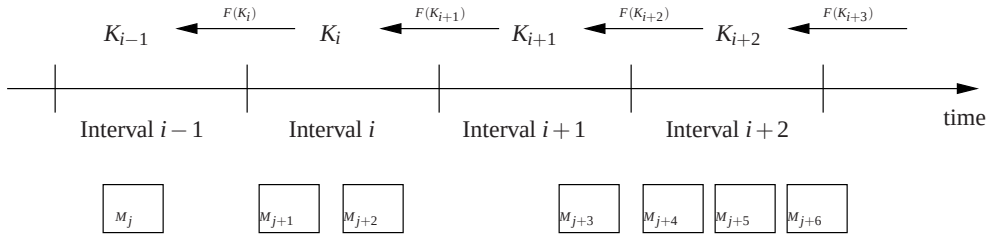


Figure 1: At the top of the figure is the one-way key chain (using the one-way function F). Time advances left-to-right. At the bottom of the figure, we can see the messages that the sender sends in each time interval. For each message, the sender uses the current time interval key to compute the MAC of the message.

authentication, and 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 keys, by selecting key K_N of interval N at random, and by repeatedly applying a one-way hash function F to derive earlier keys. A cryptographic hash function, such as SHA-1 [27], offers the required properties. The sender assigns keys sequentially to time intervals (one key per time interval). The one-way chain is used in the reverse order of generation, so any key of a time interval can be used to derive keys of previous time intervals. For example, assuming a disclosure delay of 2 time intervals, key K_i will be used to compute MACs of broadcast messages sent during time interval i, but disclosed during time interval $i+2$. The sender defines a disclosure delay for keys, usually on the order of a few time intervals. The sender publishes the keys after the disclosure time.

- The sender attaches a MAC to each message, computed over the data, using the key for the current time interval. Along with the message, the sender also sends the most recent key that it can disclose. In the example of Figure 1, the sender uses key K_{i+1} to compute the MAC of message M_{j+3}, and publishes key K_{i-1} assuming a key disclosure delay of two time intervals.

- Each receiver that receives the message 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 message. In the example of Figure 1, when the receiver gets message M_{j+3}, it needs to verify that the sender did not yet publish key K_{i+1}, by using the loose time synchronization and the maximum time synchronization error Δ. If the receiver is certain that the sender did not yet reach interval $i+3$, it knows that key K_{i+1} is still secret, and it can buffer the packet for later verification.

- 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 messages that were sent in the time interval of the disclosed key. Assuming the receiver knows the authentic key K_{i-2}, it can verify the authenticity of key K_{i-1} by checking that $F(K_{i-1})$ equals K_{i-2}. If K_{i-1} is authentic, the receiver can verify the authenticity of buffered packets sent during time interval $i-1$, since they were authenticated using key K_{i-1} to compute the MAC.

One-way chains have the property that if intermediate keys are lost, they can be recomputed using later keys. So, even if some disclosed keys are lost due to packet loss or jamming attacks, a receiver can recover the key from keys disclosed later and check the authenticity of earlier messages.

Along with each message M_i, the sender broadcasts the μTESLA authentication information. The broadcast channel may be lossy, but the sender would need to retransmit with an updated MAC key. Despite loss, each receiver can authenticate all the messages it receives.

3.2 Reducing Verification Overhead of μTESLA

Even though μTESLA provides a viable solution for broadcast authentication in sensor networks, many challenges still remain. We describe the remaining challenges below and propose extensions and new approaches to address these challenges.

Some applications broadcast messages infrequently at unpredictable times and the receivers may need to authenticate messages immediately. For example, a fire alarm event is infrequent and needs to be quickly distributed and authenticated. Unfortunately, when messages are infrequent, due to the one-way chain approach to verify the authenticity of keys, a receiver may need to compute a long chain of hash values in order to authenticate the key which could take several tens of seconds for verification. Such verification delays the message authentication significantly and may consume significant computation and energy resources. This approach also introduces a Denial-of-Service (DoS) attack: an attacker sends a bogus key to a receiver, and the receiver spends several thousands of one-way function computations (and several seconds) to finally notice that the sent key was incorrect.

One approach is to periodically release μTESLA keys and hence the work for verification of an infrequent message would be distributed over time. However, this approach wastes energy for periodic broadcast of μTESLA keys. In the same vein, a sender can publish several keys in a packet to reduce the effect of DoS attacks by requiring a receiver to perform a small number of one-way function computations to incrementally authenticate each key of the one-way chain. An advantage of this approach is that it makes the DoS attack described above less attractive to an attacker, as a receiver would need to follow the one-way chain for a short interval only to detect a bogus key.

Another approach to counteract the slow and expensive verification problem is to use a Merkle hash tree [24] instead of a one-way chain to authenticate μTESLA keys. This approach has been suggested in another context [13]. For N keys, the tree has height $d = \log_2(N)$ and along with each message, the sender sends d values to verify the key. Despite the logarithmic communication cost, this is still too large for most sensor networks: consider a network where we switch to a different hash tree every day, and we need a

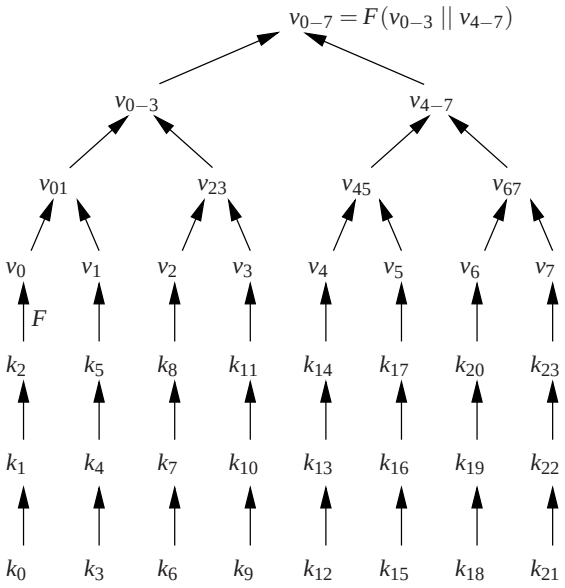


Figure 2: Hash tree constructed over one-way chains of μTESLA keys.

key resolution of 1 second. The 86,400 keys that we need in one day require a tree of height 17. Assuming a hash output of 10 bytes, the sender would need to consequently add 170 bytes to each message for authentication (17 nodes at 10 bytes each). This is far too much for most sensor networks, where nodes typically communicate with messages shorter than 100 bytes. Splitting the load up into two messages is not a viable approach, because of the usually high packet loss rates in sensor networks. The receiver would only need to compute $O(\log(N))$ operations for verification, 17 hash function computations in our example which requires around 17ms on current sensor nodes.

To reduce the bandwidth overhead, we design a different approach that achieves lower message size at the cost of higher verification computation. our approach is to combine one-way chains with hash trees. Consider the structure that Figure 2 shows. We construct a hash tree over short one-way chains. If each one-way chain has a length of k, the verification cost is expected to be $k/2 + \log(N/k)$ (it is at most $k + \log(N/k)$), and the communication cost is $\log(N/k)$. For a given upper bound on the verification time, we can thus minimize the communication overhead. Consider an upper bound on the verification time of approximately 500ms. We can set $k = 2^9 = 512$, thus the hash tree will have 8 levels, requiring 80 bytes per packet, making this an attractive approach for many applications.

An alternative approach would be to construct a hash tree over the one-way key chain, where the every k'th key will be a leaf node of the hash tree (for example, in Figure 2, the value k_0 would be derived from the previous leaf node $k_0 = F(v_1)$). The advantage of this approach is that a sender would not need to send the hash tree values along with a message, as a value can be authenticated by following the one-way chain to the last known value. However, if the sender did not send out any message during an extended time period, that authentication would be computationally expensive and thus the sender can choose to also send the hash tree nodes along for fast verification. This approach would also prevent DoS attacks since the verification is very efficient.

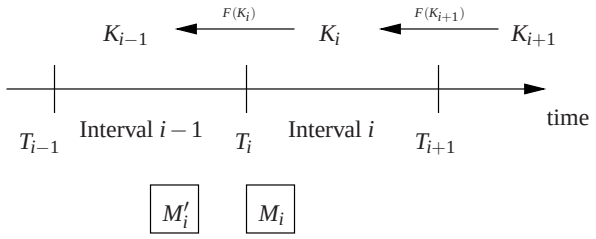


Figure 3: This figure shows authentication of one message in the RPT protocol. Message $M_i' = \langle MAC_{K_i}(M_i)\rangle$, and message $M_i = \langle M_i, K_i\rangle$.

3.3 RPT: Authenticating Messages Sent at Regular and Predictable Times

As described in our taxonomy in Section 2, one additional property in the design space of broadcast authentication is to authenticate asynchronous messages sent at irregular and unpredictable times. All protocols described so far can achieve this property. However, if we were to remove this requirement, new possible approaches exist that can only authenticate messages sent at regular and predictable times, yet satisfy all of the other cardinal properties defined in our taxonomy. In this section, we introduce our design of one such protocol called RPT, a modification of the μTESLA protocol.

In practice, many protocols send synchronous messages at regular and predictable times. The plaintext of these messages are often known by the sender a priori. In particular, messages containing meta-data are especially well-suited for this type of communication. For example, a base-station often performs key update or time re-synchronization at a preset time of day. In these examples, the sender knows exactly what message needs to be sent at a particular time, but the protocol dictates that such messages cannot be sent until a pre-specified time.

Consider an application that broadcasts a message every day at noon to all nodes. If we use standard μTESLA with one key per

day, it would take one day to authenticate the message, since the receivers would need to wait for the disclosed key one day later. On the other hand, if we use many keys, for example, one key per second, it would require 86,400 keys per day (not using the optimization we presented in the previous section), and a sensor node would require an expected time of 43 seconds to verify the authenticity of the key. Hence, if messages are sent at very regular time intervals, we can streamline μTESLA to immediately authenticate these messages.

The RPT protocol (Regular-Predictable TESLA) achieves immediate authentication for messages sent at regular and predictable times. Consider a message that needs to be sent at times $T_i = T_0 + i \cdot D$. The sender creates a one-way key chain, and assigns one key to each time interval of duration D. We assume that the sender knows the content of the message M_i to be broadcast at time T_i by time $T_i - \delta$, where δ is the maximum network broadcast propagation delay plus the maximum time synchronization error. At time $T_i - \delta$, the sender broadcasts message $\langle MAC_{K_i}(M_i)\rangle$, and at time T_i the sender broadcasts $\langle M_i, K_i\rangle$. As soon as the receiver receives the first message, it needs to verify the safety condition that key K_i is still secret, given its current time and the maximum time synchronization error. When receiving the second message, the receiver first verifies the key K_i. If the key is correct it verifies the MAC, and if the MAC is correct it is assured that M_i is authentic. Note that this approach does not exhibit any authentication delay, as the receiver can immediately authenticate M_i immediately after reception.

At first glance, it may appear that RPT is susceptible to a denial-of-broadcast attack, where an attacker sends a large number of forged MACs around the time the legitimate is sent out. This problem had been studied and addressed in previous work [16]. However, it is not easy to evaluate how well this works in practice.

4. BROADCAST AUTHENTICATION WITH ONE-TIME SIGNATURES

Another way to achieve asymmetric authentication is through the use of one-time signatures. A one-time signature is much faster to generate and verify than general purpose signatures, but the private key associated with the signature can be used to sign only a single message, otherwise the security degrades and an attacker could forge signatures. Unlike μTESLA, time synchronization is not necessary and authentication is immediate. Moreover, one-time signatures achieve non-repudiation in addition to authentication, which enables a node to buffer a message and retransmit it later. The receiver of the retransmitted message can still authenticate the message.

One-time signatures are advantageous in applications with infrequent messages at unpredictable times, as they do not add computation to the receiver based upon the time at which the message is received. This makes them resilient to many forms of DoS attacks. We now present an overview of one-time signatures, and then present our LEA broadcast authentication protocol for authentication of low-entropy messages in Section 4.2.

4.1 One-Time Signatures Overview

The Merkle-Winternitz signature was first drafted by Merkle [25, 26], and was later also used by Even, Goldreich, and Micali [8], and more recently also by Rohatgi for efficient stream authentication [35]. We briefly describe the basic principle of the Merkle-Winternitz signature.

A Merkle-Winternitz signature relies on efficient one-way functions to construct a DAG (directed acyclic graph) to encode a signature. Each edge between two vertices ($v_1 \rightarrow v_2$) in the graph represents an application of the one-way function, where the value of the end node is the result of the one-way function applied to the beginning node ($v_2 = F(v_1)$, where F represents the one-way function). End nodes with multiple incoming edges take on the value of the hash of the concatenation of predecessor nodes. The initial values of the graph represent the private key, and the final value represents the public key.

To achieve a secure one-time signature, the property of the signature encoding is that an attacker would have to invert at least one one-way function to sign any other value (i.e., forge a signature).

We now discuss an example of a signature graph and signature encoding. Figure 4(a) depicts the one-time signature. A one-way hash chain of length 4 can be used to encode the values $0-3$. For this *signature chain*, we will use the convention that the 1st value s_3 in the chain encodes the value 3, the second 2, etc.

The signer derives the value s_3 from a randomly generated private key K_{priv} by using a Pseudo-Random Function (PRF), e.g., $s_3 = PRF_{K_{priv}}(0)$.[3] To prevent signature forgery (as we will explain later), the sender also creates a *checksum chain* $c_0 \ldots c_3$, deriving value c_0 also from the private key, e.g., $c_0 = PRF_{K_{priv}}(1)$, and again using the one-way function to derive the other values, e.g., $c_1 = F(c_0)$. The application of the one-way function on s_0 and c_3 forms the public key: $K_{pub} = F(s_0 \mid\mid c_3)$. To sign value i, where $0 \leq i \leq 3$, the signer uses values s_i and c_i as the signature.

To verify the signature s_i and c_i, the receiver follows the one-way chains and recomputes the public key as follows, with $F^0(x) = x$:

$$K_{pub} = F(F^i(s_i) \mid\mid F^{3-i}(c_i))$$

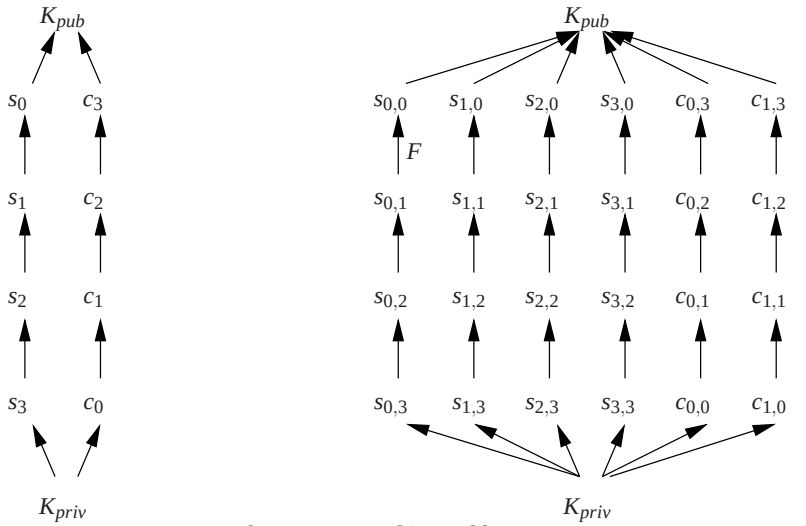
A signature is correct if the recomputed value matches the public key For example, consider a signature on value 2: s_2 and c_2. To verify, the receiver checks that $K_{pub} = F(F(F(s_2)) \mid\mid F(c_2))$.

An attacker who wishes to forge a signature is forced to invert at least one one-way function (since the indices of the checksum chain run in direction opposite to the signature chain). Assuming the one-way function is secure, an attacker cannot invert the function to forge a signature, hence, the signature is secure. In practice, we can use a secure cryptographic hash function for our one-way function, but for increased efficiency we use a block cipher in hash mode, for example the commonly used Matyas-Meyer-Oseas mode [22].

Using two chains achieves a secure one-time signature, but does not scale well to sign a large number of bits. If we use two chains, a signature on 32 bits would require a chain 2^{32} values long, which has a very high overhead to generate and verify. Instead, if more than one chain is used, each chain can encode some number of bits of the signature. For example, one could encode an 8 bit number by using four chains of length 4 to encode two bits in each chain. The public key is derived from the last value on all the chains. However, in this scheme, we would still need an additional 4 chains of length 4 to encode the values in the opposite direction to prevent forgeries.

The Merkle-Winternitz signature reduces the number of checksum chains, in that the redundant checksum chains do not encode the actual value, but instead encode the sum of the values on the signature chains. As explained in detail by Merkle [25,26], the checksum chain encodes the sum of all values in the signature chains. Assuming k signature chains that sign m bits each, the maximum sum would be $k \cdot (2^m - 1)$, thus the checksum chains would encode

[3]We use a block cipher to implement the PRF efficiently. A block cipher is a good PRF as long as we do not use the PRF to compute more than $O(\sqrt{2^n})$ operations with the same key, where n is the blocksize in bits. Since we only perform a few operations, the block cipher is a secure and efficient PRF.



(a) Simple one-time signature to sign 2 bits.

(b) Merkle-Winternitz one-time signature. This construction can sign 8 bits.

Figure 4: This figure illustrates the Merkle-Winternitz one-time signature.

$\log_2\lceil k\cdot(2^m-1)\rceil$ bits, providing for a significant savings. This approach still ensures that an attacker would have to invert at least one one-way function to forge a signature.

Using signature chains with 4 values, a signature on n bits will then require $n/2$ signature chains. Since each chain encodes up to the value 3, the checksum chain at most needs to encode the value $(n/2)*3$ as the total sum; thus, the checksum chains need to sign $\log_2(n/2*3)$ bits. If we also use checksum chains with 4 values, each checksum chain can again sign 2 bits and we need $\lceil\log_2(n/2*3)/2\rceil$ checksum chains. Figure 4(b) shows an example of such a signature for signing 8 bits. Since the four signature chains can at most encode the number 3, the total sum is at most $4*3=12$. Thus we only need 2 additional checksum chains to encode the 4 bits. Again, the indices in the checksum chain run opposite to the indices in the signature chain, to ensure that an attacker would have to invert at least one one-way function to forge a different signature.

For the specific case of signing 80 bits, researchers suggest using chains of length 16 to encode 4 bits per chain [35]. Thus, we need $20=80/4$ signature chains, and the checksum chains would need to encode at most values $0\ldots300\,(=20\cdot15)$, which will require 9 bits, which again requires 3 checksum chains (where the third chain only requires 2 values to sign a single bit).[4]

We now compute the computation overhead of signature verification. On average, signature verification requires following half the signature chains, which requires 8 one-way function computations. In the case of signing 80 bits with 20 signature chains, this will result in 160 one-way function computations. On average, the checksum chains require 16 one-way function computations, adding up to a total of 176 computations.

4.2 LEA: Authentication of Low-Entropy Messages

If messages have high entropy, the one-time signature is still quite large in size. For example, if messages have 80 bits or more of entropy, the signer can hash the message before signing it. Using the construction we discussed in Section 4.1, signing an 80-bit hash value would yield a 230 bytes signature (or 184 bytes if we assume 8 byte long hash chain values). Unfortunately, this is still too large for current sensor networks.

However, for messages with lower entropy, one-time signatures can be very effective. We thus present the LEA (Low-Entropy Authentication) protocol. The LEA protocol is based on Merkle-Winternitz one-time signatures, and periodically pre-distributes one-time public keys to receivers, and the sender uses the corresponding private keys to sign messages.

The Merkle-Winternitz one-time signature is efficient for signing small numbers of bits. For example, assuming chains of length 16, to sign a message of n bits, we would need $n/4$ signature chains. Thus we need to encode $\log_2(n/4*15)$ bits in the checksum chains, hence requiring $\lceil\log_2(n/4*15)/4\rceil$ additional checksum chains. For signing 8 bits, the signature would require 2 signature chains and 2 additional checksum chains to encode the sum ranging from $0\ldots30$, which would require 32 bytes assuming 8 byte values. Since communication cost is a premium, we could use a single checksum chain of length 30 to encode the checksum, thus saving 8 bytes. Hence, the total size of the authentication information would be 24 bytes.

Since the size of the signature depends on the number of bits being signed, this method is preferable for situations where the message is a simple time critical command, such as an alarm, or a preset command. For example, to sign 128 different commands, we would only need one signature chain with 16 values, one signature chain with 8 values, and one checksum chain with 22 values. Assuming 8 byte values, the total signature length is 24 bytes.

In some applications it may be possible to use a lossy compression algorithm to compress and quantize the data for the signature. This would allow the message to contain uncompressed data, but the attacher would only be able to change the message to a small degree. This could be helpful in commands which set the sensitivity of a motion sensor and the administrator is willing to allow a small error in the sensitivity which is actually received on the device.

One of the main challenges of using one-time signatures is to dis-

[4] We could also use 2 signature chains with 18 values each, as $18^2=324$, saving one checksum chain.

tribute one authentic public key for each signature to the receivers. Without an authentic public key an attacker could inject it's own public key and one-time signatures. This problem is easier than the original problem of general broadcast authentication because the public keys can be distributed far ahead of time at a predictable time.

There are several methods by which this may be achieved. The simplest would be to distribute a set of k public keys to each receiver at bootstrap and these keys would be usable for the first k messages. If the lifetime of the devices compared to k is small, then the devices will not have to be re-bootstrapped.

In general, the number of total messages is unknown. Thus, we design a mechanism to efficiently replenish authentic public keys after their use. We leverage the RPT protocol for this purpose. Nodes store a number of authentic public keys. The sender uses up one one-time signature (or one private key) per message it broadcasts. With this approach, all receivers can immediately authenticate the message. Periodically, the sender sends a RPT message at a regular time with new one-time public keys to replenish the used-up public keys at receivers. Since each public key is only 10 bytes long, this is an efficient approach.

4.3 Chaining Merkle-Winternitz Public Keys

The above scheme illustrates an effective way to use μTESLA in conjunction with Merkle-Winternitz signatures to provide fast and efficient authentication. The only drawback of using the Merkle-Winternitz one-time signature is that the public key can only be used once. Therefore when a μTESLA authenticated message is sent at the beginning of the day authenticating k Merkle-Winternitz public keys, the sender and receiver are limited to only being able to authenticate k messages that day. The tradeoff is that choosing a large k uses up receiver memory resources.

To circumvent this problem, rather than sending a fixed number of messages per interval, the public keys can be chained together in such a way that if more messages are needed they can be sent to the receiver and authenticated immediately.

In this approach, the sender generates a large number of public and private keys for one-time signatures, labeling the public keys $P_0, P_1, \ldots, P_n$. These public keys are then combined, such that verification of one signature will automatically authenticate the public key of the next signature:

$V_0 = P_0$
$V_1 = H(P_1 \,||\, V_0)$
...
$V_i = H(P_i \,||\, V_{i-1})$
...
$V_n = H(P_n \,||\, V_{n-1})$

In this approach, the sender only needs to send the value V_n authenticated with μTESLA. The sender subsequently uses the private key that corresponds to the public key P_n to sign a message, and sends value V_{n-1} along with the message. From the signature, the receiver can compute the public key P_n, and together with the value V_{n-1} the receiver can authenticate the public key and V_{n-1} based on the trusted value V_n. Now that the receiver trusts value V_{n-1}, the next public key P_{n-1} can be authenticated in the same way.

This approach has the drawback that the message to be authenticated also needs to carry the value V_{n-1} increasing the message size by 8–10 bytes, and that message loss prevents later messages to be authenticated. We propose to use a hybrid approach: send k public keys authenticated with RPT each day, along with one value V_n. If the sender needs to send more than k authenticated messages, it can then use the chained public keys after the first k messages.

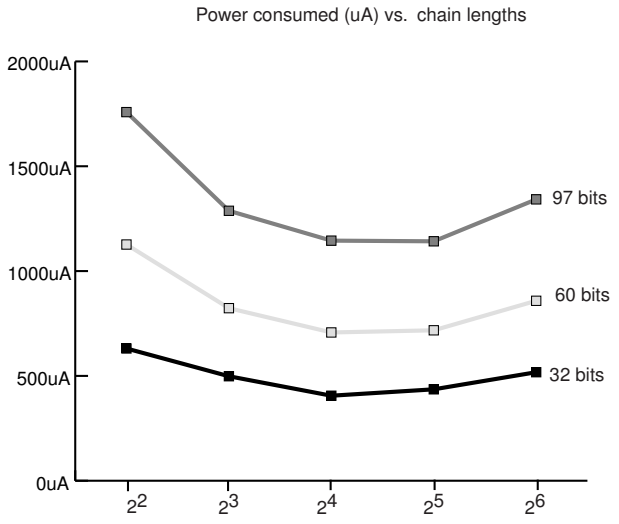


Figure 5: The power consumption for an MSP430 sensor node receiving and validating Merkle-Winternitz signatures for varying signature chain lengths.

5. IMPLEMENTATION AND PERFORMANCE EVALUATION

Figure 5 illustrates the amount of energy required for using a Merkle-Winternitz signature for signing 32 bits, 60 bits, and 97 bits. In this example, the sensor is an 16-bit TI MSP430 processor running at 1 MHz, which can compute an 8-byte hash in approximately 5ms using RC5. This processor uses up 0.28 μA per ms, and 3.8 μA per byte received. Shown are the overall power consumption for five different chain lengths, 2^2, 2^3, 2^4, 2^5, 2^6, and 2^7. Table 2 shows the power consumption, validation times, and communication overhead for signing 60 bits with varying length chains.

We implemented the PRF using the Helix stream cipher [9]. Unlike RC5, this cipher is not patented. It also features an efficient MAC construction which we use in our implementation of μTESLA. The PRF is computed by using the input to the PRF as the key in encryption mode, and using the keystream as the output of the PRF. In this implementation, it takes about 8 ms to compute an 8-byte PRF. Since the signature generation requires comparable amount of computation as verification, generation of a 64-bit signature takes about 1.2 seconds and verification takes about 1 second in our un-optimized implementation. However, in this scheme, the public keys are generated in advance, so the sender must compute twice as many hashes because it must recompute the hashes when he wishes to actually compute a signature instead of simply generating the public key. This still makes it feasible for a sensor-node to act as the base station in our implementation, but generating a large amount of public keys becomes costly. The implementation is about 4k in size, 2k for the Helix assembly code, and 2k for the Merkle-Winternitz code (with code for both generation and validation).

6. RELATED WORK

The μTESLA protocol is a viable mechanism for broadcast authentication in sensor networks [31]. Unfortunately, this approach introduces an authentication delay and thus does not provide immediate authentication of messages which is necessary in applications

	2^2	2^3	2^4	2^5	2^6	2^7
Power-cons (μA)	1126.7	823.1	707.2	717.5	858.3	1163.2
Auth-time (ms)	332.5	442.5	680.0	1042.5	1762.5	2960.0
Overhead (bytes)	272	184	136	112	96	88

Table 2: Efficiency for signing a 60 bit value using Merkle-Winternitz one-time signature.

with real-time requirements. Moreover, the μTESLA approach has some denial-of-service vulnerabilities, which we address in this paper.

Liu and Ning subsequently improved the efficiency of bootstrapping new clients, using multiple levels of one-way key chains [20]. This work also discussed the DoS attack explained in Section 3.2. Liu et al. also outlines a potential approach to authenticate commitment messages with Merkle hash trees [19].

Several researchers have investigated the use of asymmetric cryptographic techniques in sensor networks. Unfortunately, the overhead is too high to warrant use of such techniques for per-packet broadcast authentication. Such schemes were discussed in Section 2 in the context of protocols with high computation overhead.

7. CONCLUSION

We have studied viable and efficient solutions for efficient broadcast authentication in sensor networks. This problem is challenging due to the highly constrained nature of the devices and the unpredictable nature of communication in many environments. Since the authentication of broadcast messages is one of the most important security properties in sensor networks, we need to study viable approaches for a variety of settings. We establish a set of properties of broadcast authentication: security against compromised nodes, low computation and communication cost, immediate authentication (with no receiver delay), authentication of unpredictable messages with high entropy, and robustness to packet loss. We present a viable protocol for each case where we relax one property, and pose the open challenge to find a protocol that satisfies all properties.

8. REFERENCES

[1] D. Boneh, G. Durfee, and M. Franklin. Lower bounds for multicast message authentication. In *Advances in Cryptology — EUROCRYPT '01*, pages 434–450, 2001.

[2] M. Brown, D. Cheung, D. Hankerson, J. Lopez Hernandez, M. Kirkup, and A. Menezes. PGP in constrained wireless devices. In *Proceedings of USENIX Security Symposium*, August 2000.

[3] 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.

[4] J. Deng, R. Han, and S. Mishra. A performance evaluation of intrusion-tolerant routing in wireless sensor networks. In *Proceedings of IEEE Workshop on Information Processing in Sensor Networks (IPSN)*, April 2003.

[5] J. Deng, C. Hartung, R. Han, and S. Mishra. A practical study of transitory master key establishment for wireless sensor networks. In *Proceedings of the First IEEE/CreateNet Conference on Security and Privacy for Emerging Areas in Communication Networks (SecureComm)*, 2005.

[6] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained network time synchronization using reference broadcasts. In *Proceedings of Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.

[7] Jeremy Elson and Kay Römer. Wireless sensor networks: A new regime for time synchronization. In *Proceedings of Workshop on Hot Topics In Networks (HotNets-I)*, October 2002.

[8] S. Even, O. Goldreich, and S. Micali. On-line/off-line digital signatures. In *Advances in Cryptology — CRYPTO '89*, volume 435, pages 263–277, 1990.

[9] Niels Ferguson, Doug Whiting, Bruce Schneier, John Kelsey, Stefan Lucks, and Tadayoshi Kohno. Helix: Fast encryption and authentication in a single cryptographic primitive. In *Proceedings of the International Workshop on Fast Software Encryption (FSE 2003)*, 2003.

[10] V. Gupta, M. Millard, S. Fung, Y. Zhu, N. Gura, H. Eberle, and S. C. Shantz. Sizzle: A standards-based end-to-end security architecture for the embedded internet. In *Proceedings of the Third IEEE International Conference on Pervasive Computing and Communication (PerCom)*, 2005.

[11] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David E. Culler, and Kristofer S. J. Pister. System architecture directions for networked sensors. In *Proceedings of Architectural Support for Programming Languages and Operating Systems (ASPLOS IX)*, pages 93–104, 2000.

[12] Lingxuan Hu and David Evans. Secure aggregation for wireless networks. In *Workshop on Security and Assurance in Ad hoc Networks*, January 2003.

[13] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Packet leashes: A defense against wormhole attacks in wireless networks. In *Proceedings of IEEE INFOCOM*, April 2003.

[14] J. M. Kahn, R. H. Katz, and K. S. Pister. Mobile networking for smart dust. In *Proceedings of ACM/IEEE Conference on Mobile Computing and Networking (MobiCom)*, August 1999.

[15] C. Karlof, N. Sastry, and D. Wagner. TinySec: A link layer security architecture for wireless sensor networks. In *ACM SenSys*, November 2004.

[16] Chris Karlof, Naveen Sastry, Yaping Li, Adrian Perrig, and J. D. Tygar. Distillation codes and applications to dos resistant multicast authentication. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS)*, November 2004.

[17] Chris Karlof and David Wagner. Secure routing in wireless sensor networks: Attacks and countermeasures. In *Proceedings of First IEEE International Workshop on Sensor Network Protocols and Applications*, May 2003.

[18] A. Lenstra and E. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14(4):255–293, 2001.

[19] D. Liu, P. Ning, S. Zhu, and S. Jajodia. Practical broadcast authentication in sensor networks. In *Proceedings of The 2nd Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*, November 2005.

[20] Donggang Liu and Peng Ning. Efficient distribution of key

chain commitments for broadcast authentication in distributed sensor networks. In *Proceedings of Network and Distributed System Security Symposium (NDSS)*, pages 263–276, February 2003.

[21] David Malan, Matt Welsh, and Michael Smith. A public-key infrastructure for key distribution in TinyOS based on elliptic curve cryptography. In *Proceedings of IEEE International Conference on Sensor and Ad hoc Communications and Networks (SECON)*, October 2004.

[22] S. Matyas, C. Meyer, and J. Oseas. Generating strong one-way functions with cryptographic algorithm. *IBM Technical Disclosure Bulletin*, 27:5658–5659, 1985.

[23] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[24] R. Merkle. Protocols for public key cryptosystems. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 122–134, April 1980.

[25] R. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology — CRYPTO '87*, pages 369–378, 1988.

[26] R. Merkle. A certified digital signature. In *Advances in Cryptology — CRYPTO '89*, pages 218–238, 1990.

[27] National Institute of Standards and Technology (NIST), Computer Systems Laboratory. Secure Hash Standard. Federal Information Processing Standards Publication (FIPS PUB) 180-2, February 2004.

[28] A. Perrig. The BiBa one-time signature and broadcast authentication protocol. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, pages 28–37, November 2001.

[29] 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.

[30] A. Perrig, R. Canetti, J. D. Tygar, and D. Song. The TESLA broadcast authentication protocol. *RSA CryptoBytes*, 5(Summer), 2002.

[31] Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, and J. D. Tygar. SPINS: Security protocols for sensor networks. In *Proceedings of ACM Conference on Mobile Computing and Networks (MobiCom)*, pages 189–199, 2001.

[32] Bartosz Przydatek, Dawn Song, and Adrian Perrig. SIA: Secure information aggregation in sensor networks. In *Proceedings of the First ACM International Conference on Embedded Networked Sensor Systems (SenSys 2003)*, pages 255–265, November 2003.

[33] Leonid Reyzin and Natan Reyzin. Better than BiBa: Short one-time signatures with fast signing and verifying. In *Proceedings of Conference on Information Security and Privacy (ACISP)*, July 2002.

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

[35] 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. ACM Press, November 1999.

[36] F. Ye, H. Luo, S. Lu, and L. Zhang. Statistical en-route filtering of injected false data in sensor networks. In *Proceedings of IEEE INFOCOM*, March 2004.

[37] S. Zhu, S. Setia, S. Jajodia, and P. Ning. An interleaved hop-by-hop authentication scheme for filtering false data in sensor networks. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 259–271, May 2004.