Secure and Efficient Broadcast Authentication in Wireless Sensor Networks

Taekyoung Kwon, *Member, IEEE,* and Jin Hong, *Non-member, IEEE,*

Abstract—Authenticated broadcast, enabling a base station to send commands and requests to low-powered sensor nodes in an authentic manner, is one of the core challenges for securing wireless sensor networks. μTESLA and its multi-level variants based on delayed exposure of one-way chains are well known valuable broadcast authentication schemes, but concerns still remain for their practical application. To use these schemes on resource-limited sensor nodes, a 64-bit key chain is desirable for efficiency, but care must be taken. We will first show, by both theoretical analysis and rigorous experiments on real sensor nodes, that if μTESLA is implemented in a raw form with 64-bit key chains, some of the future keys can be discovered through time memory data tradeoff techniques. We will then present an extendable broadcast authentication scheme called X-TESLA, as a new member of the TESLA family, to remedy the fact that previous schemes do not consider problems arising from sleep modes, network failures, idle sessions, as well as the time memory data tradeoff risk, and to reduce their high cost of countering DoS attacks. In X-TESLA, two levels of chains that have distinct intervals and cross-authenticate each other are used. This allows the short key chains to continue indefinitely and makes new interesting strategies and management methods possible, significantly reducing unnecessary computation and buffer occupation, and leads to efficient solutions to the raised problems.

Index Terms—Security, broadcast authentication, time-memory-data tradeoff, wireless sensor networks.

1 Introduction

Technological advancement in large scale distributed networking and small sensor devices has led to the development of wireless sensor networks with numerous applications [1]. Sensor nodes are usually constrained in their computation, communication, storage, and energy resources for economical reasons, but need security functions since they are deployed in unattended or even hostile environments. The high risk of physical attacks and the limited capabilities of sensor nodes make it difficult to apply traditional security techniques to wireless sensor networks, posing new challenges [29].

Authenticated broadcast, enabling a base station to send authentic messages to multiple sensor nodes, is one of the core challenges [20], while even the broadcast by nodes is an important topic in wireless sensor networks [7], [21], [25]. For the purpose, digital signatures (public-key) are not very useful in a resource limited environment, while naïve use of HMAC (secret-key) does not work either, as node capture can lead to a key compromise. μTESLA and its multi-level variants [18], [19] based on TESLA [26], [27], use a one-way chain practically [13] under a loose time synchronization assumption. The sender attaches a MAC (Message Authentication Code) to each packet, computed using a key from the chain in reverse order. The keys are exposed after a certain time delay. The receiver buffers the received packet until the corresponding key is disclosed and verifies the MAC, after authenticity of the key itself has been verified by following through the chain.

- *T. Kwon is with the Department of Computer Engineering, Sejong University, Seoul, 143-747, Korea. E-mail: tkwon@sejong.ac.kr.*
- *J. Hong is with the Department of Mathematical Sciences and ISaC, Seoul National University, Seoul, 151-747, Korea. E-mail: jinhong@snu.ac.kr.*

1.1 Motivation (and Problems)

μTESLA and its variants are designed to be practical, but significant concerns still remain.

1.1.1 64-bit key chain

A short 64-bit key chain is desirable for efficiency in resource-limited sensor nodes, but care must be taken, even with short time intervals. As we show, if the chain is generated in a straightforward manner, TMD (Time Memory Data) tradeoff techniques can be applicable, leading to discovery of future keys.

1.1.2 Sleep mode or network failure

If sensor nodes go into a sleep mode or key disclosure messages are lost frequently, μTESLA may force heavy key computation to be done at once on sensor nodes for chain verification, during which incoming packets get dropped. If CDMs (Commitment Distribution Messages) are missing, multi-level μTESLA makes nodes wait and buffer for the long interval of upper levels, during which incoming packets are dropped due to the buffer limit.

1.1.3 Idle sessions

Even for idle sessions with no broadcasts, μTESLA forces chain computation for sensor nodes. Key disclosure messages should be broadcast constantly or heavy computation needs to be done later. Multi-level μTESLA needs CDMs to be broadcast for higher levels, with the number of CDMs increasing with the number of levels.

1.1.4 Extended lifetime

With node malfunctions and premature power exhaustion, there are needs for node additions [1] or rechargeable sensor nodes [15]. Thus, the lifetime of a network may extend beyond that of each node. As noted in [18], [19], lifetime extension was not clearly considered in μTESLA. Multi-level μTESLA should also fix the lifetime.

1.1.5 DoS attacks

To resist DoS attacks, multi-level μTESLA requires many CDMs to be distributed for longer intervals. Its DoS tolerant version needs sufficiently large buffers on sensor nodes for random selection of received CDMs. The DoS resistant version requires CDMs to be received stably along with a larger packet and additional hash function.

1.2 Our Contribution (and Results)

The contribution of this paper is two-fold.

(i) We first show, both by theoretical analysis and rigorous experiment on real sensor nodes, that if μTESLA is used with parameters that are currently widely considered to be appropriate, a non-negligible number of future keys can be recovered with cryptanalytic tradeoff techniques, utilizing realistic resources. One may have tried applying tradeoff techniques to the 64-bit one-way function of the μTESLA family and found it not useful, as it does not recover the current key to be used. However, with *two non-trivial tricks*, we turn this around. We aim to invert 16-step iterations of the usual one-way chain considered, within a 40-day period on multiple targets. As a result, an adversary recovers a key to be released in the 16th of 200ms intervals afterward, from knowledge of a key most recently disclosed, and eventually generates the keys which allow him to send many authenticated messages of his choice for the duration of 14 intervals, or equivalently, 2.8 seconds. To demonstrate the effectiveness of this attack, we construct the Hellman table on a modern PC, and launch the attack in our test-bed environment using real sensor nodes such as Berkeley Mica-Z and Tmote Sky (Telos rev. B) motes. Note that we are not insisting on the insecurity of μTESLA, as a simple fix is available, but want to emphasize that care must be taken when implementing μTESLA from the literature alone.

(ii) We then present an *eXtendable* broadcast authentication scheme called X-TESLA as a new variant in which the aforementioned concerns are all considered. *(a)* Two-level[1] chains that have distinct intervals and cross-authenticate each other are used with four types of packets. This allows the chains to continue indefinitely and shorter chains to be used, leading to reduced memory and pre-computation requirements. *(b)* Promising variations are also considered. *(c)* The TMD-tradeoff attack problem is resolved without increasing the key size and staying within the bounds set by default 29-byte data payload size of TinyOS. *(d)* A specific design of one-way chains using a blockcipher is given for efficiency, i.e., without additional encryption. *(e)* The *commitment hopping strategy* and sleep mode management are proposed as well. Sensor nodes then can skip unnecessary chain computation for a *future key* later and go into a long sleep mode stably. With these methods, X-TESLA can cope with the problems coming from sleep modes, network failures, and idle sessions. *(f)* DoS attacks are resisted without requiring large buffers or strict commitment delivery guarantee, both of which were required for multi-level μTESLA. *(g)* We also conduct prototype implementation and performance analysis.

1. For those confusing X-TESLA and two-level μTESLA, we compare them briefly. In X-TESLA two level chains work in parallel for the same duration whereas the lower level chains are derived from the next upper level chains, repeatedly. In two-level μTESLA each key of the fixed upper level chain yields distinct lower chains.

1.3 Organization

The remainder of this article is organized as follows. In Section 2, we review related work on broadcast authentication for wireless sensor networks, and discuss their problems and shortcomings. In Section 3, we show how TMD-tradeoff can be applied to μTESLA with a detailed attack algorithm and also a concrete implementation result. In Section 4, we introduce an extendable broadcast authentication scheme called X-TESLA. Security and performance of X-TESLA are analyzed in Section 5. We conclude this paper in Section 6. The Appendices contain additional technical details.

2 BACKGROUND

Implementation of public-key cryptosystems is becoming possible, but still expensive [14], [22]. Energy-efficient sensor nodes are also great concerns [5]. More practically in this section, we briefly review μTESLA and its multi-level variant for moderate sensor nodes [18], [19], [28]. All these schemes are constructed without using public-key cryptography.

2.1 μTESLA

We give a short description of μTESLA, referring readers to [28] for more detail. μTESLA is a broadcast authentication mechanism for distributed sensor networks, which was adapted from TESLA [26]. In short, a delayed exposure of one-way chain is used for authentication. For this, it is required that the base station and sensor nodes be loosely time synchronized with a known maximum synchronization discrepancy bound. Unlike TESLA, which authenticates the initial packet with a digital signature, μTESLA uses only symmetric key techniques. The sender first fixes a public one-way function F and chooses a random value K_n. The one-way chain $K_i = F(K_{i+1})$ is iteratively calculated for all $n > i \geq 0$ and the last element K_0 is pre-installed in each receiver, the sensor node, as an initial *commitment*. μTESLA also provides a method for bootstrapping a new receiver through *unicasting*. Time is divided into short intervals. During the i-th interval I_i, the messages broadcast are sent with a MAC keyed with K_i. After a suitable delay, the key K_i itself is broadcast. Given a key K_i, calculating K_j for $j > i$ is expected to be infeasible, but anybody can calculate K_j for $j < i$, so it is easy to check the validity of any newly received K_i with the commitment K_0, or any other K_j satisfying $j < i$.

2.2 Multi-level μTESLA

One drawback of the single chain used in μTESLA is that there is a practical limit to its length, leading to a usage time limit. Also, the bootstrapping of a new receiver in μTESLA utilizes unicasting and hence is not scalable. Multi-level μTESLA [18], [19] solves this problem by using multiple chains in multiple levels.

Several levels of chains are used with each (except for the top) level consisting of multiple chains. The lowest level is a normal chain used for message broadcasts and usually lasts for a relatively short period. Upper levels exist to authenticate their very next lower level chains. When the lowest level chain draws to an end, the second level chain is used to authenticate the commitment for the next lowest level chain to be used. The second level broadcasts the CDM, with clear expression of i in the MAC only,

$$\mathrm{CDM}_i = i \| K_{i+2,0} \| \perp \| \mathrm{MAC}_{K_i}(i \| K_{i+2,0} \| \perp) \| K_{i-1},$$

in its i-th interval, where $\|$ denotes concatenation. Note that $\perp$ means a null value which is ignored in a basic version but will be replaced by a hash value in a DoS resistant version. The new commitment $K_{i+2,0}$ for the lowest level is authenticated with the key[2] K_i. Note that $K_{i+2,0}$, the lowest level commitment corresponding to the $(i+2)$-th second level interval, can only be verified by the sensor nodes after receiving CDM_j with $j > i$. To authenticate a new 2nd level chain commitment, the 3rd level is used, and so on. The top level is a single chain that has to last as long as the sensor network lifetime. So even though the use of multiple levels allows shortening of each chain, the total lifetime still has to be predefined.

Since CDMs are distributed within a longer time interval, DoS attacks must be considered. Multi-level μTESLA provides two variants for this. One is the DoS *tolerant* version with a random selection method, requiring large node buffers to store multiple CDMs for each level. The other is the DoS *resistant* version, and uses a hashing technique adapted from TESLA's immediate authentication method. This demands large base station storages for additional pre-computed chains, and larger payload CDMs, with hash value $h(\mathrm{CDM}_{i+1})$ replacing $\perp$.

2. The MAC key is *derived* from K_i through some pseudo-random function, but we shall ignore this throughout this paper for simplicity.

3 Time Memory Data Tradeoff Attack on μTESLA

We shall show that if μTESLA is used with parameters that are currently widely considered to be appropriate, say 64-bit keys, TMD-tradeoff techniques [3], [4], [10], [11], [12] can disclose future keys, only relying on realistic resources. Since a simple fix is possible we do not insist on the insecurity of μTESLA, but this shows that a simpleminded implementation can be broken not only theoretically but also in a real-world sense.

Target System: To keep our discussion simple, we shall fix various parameters, but no small tweaking of these parameters will make the system immune to our attack. We assume a multi-level μTESLA with 64-bit key chains created by a one-way function F. Adjusting our attack to single-level μTESLA will be straightforward. Our target system will disclose a key every 200ms with a delay of two time intervals at the lowest level and start a new chain every 1 hour. Appropriateness of this choice is explained in Appendix A.

Attack Objective: Readers with experience in the tradeoff technique will see that applying it to the one-way function F is useless, as the current key is not retrieved. So we take the non-trivial approach of having our attacker recover the key to be released in the 16th future 200ms interval (3.2 seconds later), from the key most recently disclosed. If such an undisclosed key is discovered within 200ms of obtaining the current disclosed key, from it, an attacker can generate the keys which will allow sending of authenticated messages for the duration of approximately 14 intervals (2.8 seconds). Loss of control for even a short amount of time can be devastating to the sensor network security, as the attacker may force nodes to replace all their current level keys with commitments of his choice. Once this has been done, commands from the real base station will no longer be authenticated and the attacker gains full control over the sensor network.

3.1 Attack Overview

Our attacker will work over a 40-day period. Throughout this period, on each of the 200ms intervals, he will repeatedly try to see if he can recover the key that is to be disclosed 16 steps later from the current disclosed key. The choice of 16 was taken to give the attacker enough time for commitment replacement and may be adjusted to meet theattacker's needs.

Let us write $H = F^{16}$ to denote 16-iterated applications of the one-way function F used in constructing the (bottom-level) chain. Notice that if y is the current 64-bit disclosed key and x is the key to be disclosed 16 steps later, then $y = H(x)$. So, the attacker wishes to find x, given $y = H(x)$. As H is not injective, not all such x will be the correct future key, but we shall ignore this for now. Consider the set of all keys disclosed during the 40-day period. These would consist of multiple shorter chains, each lasting one hour. After removing 15 starting[3] keys from each of these shorter chains, we name the resulting set that contains

$$D := \left(\frac{1}{0.2}\cdot 60\cdot 60-15\right)\cdot(24\cdot 40)\sim 2^{24.04} \qquad (1)$$

keys as $\hat{\mathcal{D}}$.

We shall give an algorithm which processes each of the keys from $\hat{\mathcal{D}}$, over a 40-day period, and finds a pre-image under H for some of these. The 15 keys were removed as there are no 16-step future keys for these one-way chain beginnings. The algorithm is expected to find the *correct* future key with 64.3% probability, and can process each key within 200ms of receiving it, when run on a PC.

Our choice of 40 days, which is equivalent to the choice of $D \sim 2^{24}$, and the choice of parameters $m = 2^{27}$ and $t = 2^{13}$, to appear below, may seem arbitrary. At this stage, we can only state that any choice with their product mtD approximately equal to the key space size 2^{64}, will work. Our specific choices were made so that the storage size m, pre-computation effort mt, and target count D are all within available resources. While their true meaning can only be understood after the algorithm and its analysis are understood, one may keep in mind that the succes of the attack basically relies on the birthday paradox to produce a collision between the pre-processed mt keys and the D online keys.

3.2 The Algorithm

The attack will proceed in two stages. The first will be a pre-computation phase, that produces a 2GB Hellman table. This will be used in the online phase to invert H. Section 3.4 explains why our algorithm works.

3.2.1 Table Creation

We fix the *chain length* to $t = 2^{17}$ and the *number of chains* to $m = 2^{27}$. We take a nontrivial permutation P acting on 64-bit values and define $\tilde{H} = P \circ H$. Here, P may be as simple as the reordering of the two 32-bit words constituting a 64-bit value. Its obscure purpose is explained in Appendix C.3.

Algorithm 1 Create Hellman table $\mathcal{T}$

```
1: Open an empty table $\mathcal{T}$.
2: for $0 \le i < m$ do
3:     Choose random 64-bit value $x$.
4:     $y \leftarrow x$
5:     for $0 \le j < t$ do
6:         $y \leftarrow \tilde{H}(y)$
7:     end for
8:     Add ordered pair $(x, y)$ to $\mathcal{T}$.
9: end for
10: Sort $\mathcal{T}$ according to the second components.
```

The attacker goes through Algorithm 1 to create a table containing m entries. With each entry taking 16 bytes, the created table is 2GB in size. Table creation requires $t \cdot m = 2^{40}$ applications of $\tilde{H}$ and a one-time sorting of m elements. The $\tilde{H}$ part dominates the time, which is (almost) equal to $2^{40} \cdot 16 = 2^{44}$ applications of F. This is within reach of current computational power.[4] Once this table is created, it may be used multiple times to attack multiple deployments of the system that uses the same F to create key chains.

3.2.2 Online Phase

In the second phase, the attacker attempts to invert H. Armed with the 2GB table $\mathcal{T}$, he runs Algorithm 2, processing every disclosed key $y \in \hat{\mathcal{D}}$.

Algorithm 2 Invert $H = F^{16}$

```
Require: Hellman table $\mathcal{T}$ created by Algorithm 1
1: $InvCtr \leftarrow 0$
2: for every $y \in \hat{\mathcal{D}}$ do
3:     $y' \leftarrow P(y)$
4:     for $0 \le i < t$ do
5:         if $(x', y') \in \mathcal{T}$ for some $x'$ then
6:             $x \leftarrow x'$
7:             for $0 \le j < t - i - 1$ do
8:                 $x \leftarrow \tilde{H}(x)$
9:             end for
10:            if $\tilde{H}(x) = P(y)$ then
11:                print "$x$ maps to $y$ under $H$."
12:                $InvCtr \leftarrow InvCtr + 1$
13:            else
14:                print "False alarm." (optional)
15:            end if
16:        end if
17:        $y' \leftarrow \tilde{H}(y')$
18:    end for
19: end for
20: print "$InvCtr$ keys inverted." (optional)
```

Each outer y-loop of Algorithm 2 requires $t = 2^{13}$ applications of $\tilde{H}$, disregarding false alarms. This amounts to approximately $16 \cdot t = 2^{17}$ applications of F and computing this within 200ms is equivalent to requiring $5 \cdot 2^{17}$ applications of F per second. If a key to ciphertext mapping of a 64-bit blockcipher is used to implement F, this corresponds to encryption speed of 5 MB/sec, which can easily be done on modern PCs.[5]

3.2.3 Sensor Network Control Takeover

Deferring careful explanation to Section 3.4, we state that the above algorithms are expected to return x satisfying $H(x) = y$ about 17.5 times and that there is a 64.3% chance that one of these is the correct future

3. These are the ones that would be disclosed last.

4. In our experiments, a Linux system with four AMD Opteron 875 processors (dual-core, 2.2GHz) and 32GB RAM was used, and Algorithm 1 took 6 days to complete.

5. If a very slow F is used, the attacker could set up multiple machines. For example, if 600ms is required to run a single y-loop of Algorithm 2 on a specific system, three machines would be prepared, and these would take turns processing the sequentially disclosed keys.

key. Although the attacker has no means of knowing beforehand which of the 17.5 instances is the correct key, commands can still be sent out to the sensor nodes, authenticated with his key guess. A small number of tries will go unnoticed. Explicitly, the attacker may send out the following set of commands to the sensor nodes:

(a) Repeated reconfiguration for battery consumption: `reroute`, `change_cluster`, `wakeup`.
(b) Enticing malfunction: `change_nodes`, `detach_region`, `disabled_list`, `change_param`.
(c) Network domination: `change_keys`, `change_commitment`, `change_bs`, `add_nodes`.

The `SimpleCmdMsg` format of TinyOS could be utilized. As its size is only 13 bytes, a single command fits into one packet easily. The usual 36-byte TinyOS packet takes less than 30ms to send on a 10kbps radio network, while the 39-byte ZigBee packet takes less than 10ms on a 50kbps radio network. So the above sets of commands should work even if we had a window of a single second.

3.3 Attack Implementation

We run both simulation and real world test of attacks.

3.3.1 Function Choice

Following the most commonly cited example in the related literature, our one-way function F was created from RC5 [31]. We need to be more explicit, as RC5 is a parameterized family of blockciphers, among which the most commonly used version utilizes 32-bit words, 12 rounds, and 128-bit keys. The 32-bit word implies 64-bit blocks and is suitable for us, but as the chain needs the key size to be of 64 bits, we took the 32-bit word, 12 round, 64-bit key version. The one-way function maps a 64-bit key to the 64-bit ciphertext which is an encryption of the all-zero plaintext under the given key. The swapping of two 32-bit words constituting a 64-bit value was used as permutation P.

3.3.2 Hellman Table Creation

Instead of starting each Hellman chain with a random 64-bit value, we used the numbers 0 through $2^{27} - 1$ as these initial points. As these can be written down in a 32-bit space, the total Hellman table size became 1.5GB instead of the 2GB, referred to in our discussion. This allows for the Hellman table to be loaded onto a PC's 2GB memory with ample room left for the OS. In fact, we use a Cygwin Unix emulation environment on a PC in which only 1.5GB memory is allowed, and the 1.5GB table must fit into that memory without a large loss.

3.3.3 Online Phase Simulation

Random chains corresponding to 40 days were generated, with each hour starting a new chain from a new random starting point, for a total of $960 = 24 \cdot 40$ independent chains. We simulated the online phase on our Opteron system, with the target chains distributed over the 8 cores. It took 40 hours to complete, meaning 13.3 days on a single core. As our requirement of 200ms per key processing allows this to be done over a 40-day period, this is three times faster than what we would need.

Using the same Hellman table, we did ten simulations with ten independently generated target data sets. Many *correct* 16-step future keys were obtained and we observed 80% probability of success. Details are given in Appendix B.

3.3.4 Sensor Network Application

To check the online phase in real time and to demonstrate the effectiveness of this attack, we took one of the many 1-hour chains that resulted in a correct 16-step future key and performed a test using real sensor nodes.

We first construct our μTESLA base station by placing the chosen 1-hour chain on a PC with dual AMD Opteron 244 (1.8GHz) processors and 4GB of memory. A Tmote Sky (Telos rev.B) is connected to the PC through a USB port, and is forwarded μTESLA messages through a Serial Forwarder (SF) using UART, which is then sent over a IEEE 802.15.4 radio channel. μTESLA is implemented in C with 200ms time intervals. A Berkeley Mica-Z sensor node in which the chain commitment is installed, can verify the μTESLA messages containing data and keys. We have the sensor node blink its yellow LED for verified data messages and its green LED for key disclosure messages. The Hellman table is placed on another PC to act as the attacker. Two Tmote Skys are connected to the attack PC through USB ports, so as to listen to the base station and send out forged messages. An attack program implemented in C communicates with the *Listener* and the *Sender* through UARTs, and sends out forged commands making the sensor node blink its red LED. Through our attack experiment, we were able to visually check the red LED flashing. This is the result of the attacker's forged messages, created using 14 valid keys, each corresponding to one interval. Details of the experiment are discussed in Appendix B.

3.4 Tradeoff Attack Analysis

In this section, we shall give a brief analysis of our attack algorithm. This section is somewhat technical and may be skipped by anyone that can believe that our attack succeeds with a reasonable probability and that it can be applied to most modifications of our explicit attack target. The more subtle aspects of the algorithm are discussed in Appendix C.

3.4.1 Attack Success Probability

Let us see what probability of success we can expect from our attack. We start with a small lemma, whose proof is elementary. Consider a set $\mathcal{N}$ of size N. Randomly choose and fix D distinct elements from $\mathcal{N}$ and name the set $\mathcal{D}$. Next, randomly choose H elements from $\mathcal{N}$, one at a time, with replacements, and call the collection $\mathcal{H}$. The family $\mathcal{H}$ may contain overlapping elements.

Lemma 1: Assuming $D \lll N$ and $DH \sim N$, the probability of $\mathcal{D}$ and $\mathcal{H}$ containing at least one element in common can be approximated by

$$p \sim 1 - \exp\Big(-\frac{DH}{N}\Big). \tag{2}$$

Let us apply Lemma 1 to our attack setting. The base space $\mathcal{N}$ will be the set of all possible 64-bit keys, so that $N = 2^{64}$. Next, consider the set of all online keys disclosed during the 40-day period. These would consist of multiple shorter chains, each lasting one hour. We remove 15 *ending* keys[6] from each of these shorter chains and take the resulting set as $\check{\mathcal{D}}$.[7] The number of elements D in $\check{\mathcal{D}}$ is given by equation (1), as before. These D elements may be assumed to be distinct, for, if otherwise, the key chain would repeat itself and the authentication system would fall under a more trivial attack.

Take $\check{\mathcal{H}}$ to be the family of keys appearing as input points to mapping $\tilde{H}$ applied during creation of $\mathcal{T}$. This excludes the ending points of each Hellman chain, and refers to $H = t{\cdot}m = 2^{40}$, possibly overlapping, elements.[8]

Now, a careful review of Algorithm 2 will reveal that should there be any element common to $\check{\mathcal{D}}$ and $\check{\mathcal{H}}$, it will be returned by Algorithm 2. This common element $x \in \check{\mathcal{D}}$ maps to the disclosed key $H(x) \in \hat{\mathcal{D}}$ and implies success of attack. The success probability of our attack can be calculated as

$$p \sim 1 - \exp(-1.029) \sim 0.643, \tag{3}$$

by substituting various numbers into Lemma 1.

3.4.2 Parameter Tweaks

In this subsection, we consider application of our tradeoff attack to other sensor network configurations.

(i) Shorter Disclosure Interval: Suppose the sensor network uses key disclosure interval shorter than the 200ms we have considered. This would result in a larger online target set being available to the attacker for the same (40-day) period of attack. This allows the success probability of attack to be maintained with a shorter Hellman chain. Hence the attacker can cope with the shorter time interval allotted to processing of each key. There may still seem to be one problem, as the attacker recovers the 16-interval future key and this is closer in real time than before. But a faster disclosure interval would usually mean a faster radio network, and hence the attacker would be satisfied with the shorter time available for trying out of the recovered key. Another approach the attacker may take is to attempt to recover keys further steps into the future. This would require longer pre-computation time for the same length Hellman chains and a more powerful system during the online phase. By a more powerful system, we mean that one could either use a faster processor, or let multiple processors take turns processing the target data, each for a time span longer than the disclosure interval.

(ii) Longer Disclosure Interval: If the opposite approach of using longer disclosure interval is used, the attacker has less online target data available than before. But this gives him more time to process each target data, so longer Hellman chains can be used. This will result in the pre-processing time increasing, but an increase by a small factor is well within current computational power. The attacker can also take the approach of trying to recover keys smaller steps into the future. Then the longer Hellman chains will not take longer to create.

3.4.3 Discussion

The tradeoff attack technique has been known for a long time, and this raises the question as to why delayed exposure of 64-bit one-way key chains had widely been accepted as a plausible authentication method.

Note that for a straightforward application of the original Hellman method [11] or the more widely known rainbow table method [24], a pre-computation phase consisting of about 2^{64} calculations of the one-way function is required. While no one can say for sure that this is currently impossible, it does seem to be out of reach for most organizations. Coupled with the resource constrained environment, these 64-bit one-way chain methods seem acceptable at first sight. But the Hellman method and rainbow table method deal with only a *single* target data. Our approach of trying multiple times over an extended period and being content with succeeding just once seems to have been overlooked.

The multiple target version of tradeoff attack technique we have used in this paper, applicable to any one-way function, is not new and has been developed in [3], [4], [10]. But until it was made explicit by the recent work [12], many took this to be applicable to only streamciphers in a particular way.

The main contribution of this paper concerning the weakness of current μTESLA is of pointing out that *multiple* target version of pre-computation attack is naturally applicable to the one-way chains. In doing this, the idea of looking into a 16-step composition of what would usually have been taken as the one-way function of interest was crucial. As long as succeeding even once within an extended time period is a realistic threat, there seems to be no way of using 64-bit one-way chains without *salting* them, that is, even on low-security applications.

4 X-TESLA: Secure and Efficient Broadcast Authentication Protocol

4.1 Overview of X-TESLA

Our basic idea starts from the *extendable* management of short key chains. In essence, we make two levels of

6. These are the ones that would be disclosed the earliest, and includes, in particular, the commitment.

7. The set $\check{\mathcal{D}}$ (D-check) is different from $\hat{\mathcal{D}}$ (D-hat) introduced earlier in Section 3.1.

8. For those familiar with time memory tradeoff techniques, we add the remark that our Hellman table is far from reaching the *matrix stopping rule*, and hence is free from problems due to elements being taken in chain groups and not totally independently of each other.

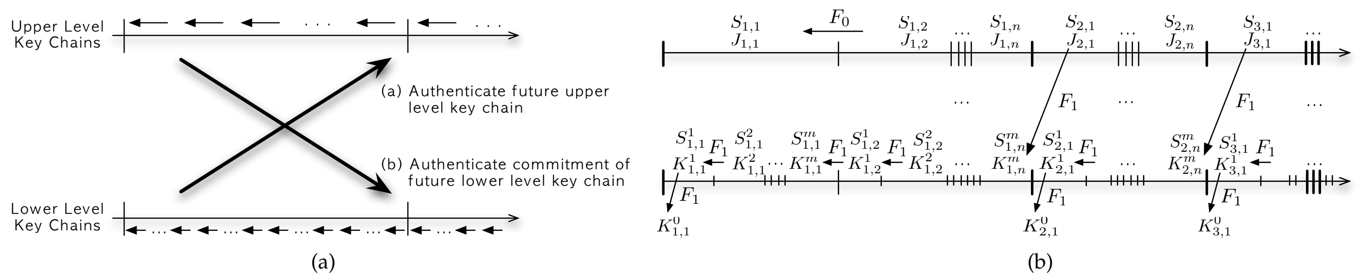

Upper Level Key Chains
(a) Authenticate future upper level key chain
(b) Authenticate commitment of future lower level key chain
Lower Level Key Chains
(a)
(b)

Fig. 1: Basic Concept of X-TESLA. (a) Cross authentication. (b) Basic flows. (An upper level chain and a lower level chain run for the same duration in parallel but with different time intervals, so that the upper level chain is composed of much smaller number of keys than the lower level chain. The last key of the lower level chain is derived from the first key of the next upper level chain.)

chains having distinct time intervals cross-authenticate each other (Fig. 1a) to provide permanently extendable chains. Our protocol X-TESLA, read either as eks TESLA or cross TESLA, stands for eXtendable TESLA. As with other TESLA variants, X-TESLA provides broadcast authentication, under the assumption that the base station and sensor nodes are loosely time synchronized with a known maximum synchronization discrepancy.

The crossing of Fig. 1a illustrates the followings.

(a) The lower level chain naturally authenticates the next upper level chain, as they are connected in a single chain by construction.

(b) Multiple *distinct* keys in the upper level chain authenticate the initial commitment of the next lower level chain repeatedly.

The repeated authentication will help in resolving problems from DoS attacks, sleeping nodes, and idle sessions.

4.2 Basic Framework of X-TESLA

4.2.1 X-TESLA chains

Two functions $F_0(\,\cdot\,,\,\cdot\,)$ and $F_1(\,\cdot\,,\,\cdot\,)$, mapping $\mathcal{K}\times\mathcal{S}$ to $\mathcal{K}$ will be used. Here, $\mathcal{K}$ denotes the key space, and $\mathcal{S}$ is the *salt* space. For each fixed $s \in \mathcal{S}$, we expect the operator $F_i(\,\cdot\,, s)$ on $\mathcal{K}$ to be one-way, even when s is known. In practice, we design the two functions with 64-bit blockciphers taking 64-bit keys and salt as plaintext in Section 4.4. The two functions may even be instantiated with the same blockcipher. Let us divide time into intervals with indices $\langle u, v\rangle$ and $\langle u, v, w\rangle$ used for the upper and lower levels, respectively. Let u index both level chains having the same durations for $u > 0$, v do intervals of each upper level chain for $0 < v \leq n$, and w divide those intervals minutely for a corresponding lower level chain for $0 < w \leq m$. Intervals themselves will be denoted as $I_{u,v}$ and $I^w_{u,v}$. We let $J_{u,v}$ and $K^w_{u,v}$ denote the corresponding upper and lower level keys. When $v = 0$ or $w = 0$, an indexed key is a commitment.

One of distinctive features of X-TESLA is the use of salt values denoted by $S_{u,v}$ and $S^w_{u,v}$, whose choice we defer to Section 4.4. These will remove TMD-tradeoff concerns by making pre-computation infeasible. After fixing each salt value, we define the upper level chain for each positive integer $u > 0$, by starting from a random seed key $J_{u,n} \in \mathcal{K}$ and recursively setting

$$J_{u,v} = F_0(J_{u,v+1}, S_{u,v+1}) \qquad (0 < v < n).$$

The resulting chain is $J_{1,1} \leftarrow J_{1,2} \leftarrow \cdots \leftarrow J_{1,n} \not\leftarrow J_{2,1} \leftarrow J_{2,2} \leftarrow \cdots$, where $\not\leftarrow$ signifies a disconnection.

For each $u > 0$ and $n \geq v > 0$, the lower level chains are constructed by recursively setting

$$K^w_{u,v} = \begin{cases} F_1(K^{w+1}_{u,v}, S^{w+1}_{u,v}) & (0 < w < m) \\ F_1(K^1_{u,v+1}, S^1_{u,v+1}) & (w = m), \end{cases}$$

starting from the seed key $K^m_{u,n} = F_1(J_{u+1,1}, S_{u+1,1})$ and ends with the commitment $K^0_{u,1} = F_1(K^1_{u,1}, S^1_{u,1})$. The resulting chain is $K^0_{1,1} \leftarrow K^1_{1,1} \leftarrow K^2_{1,1} \leftarrow \cdots \leftarrow K^m_{1,1} \leftarrow K^1_{1,2} \leftarrow \cdots \leftarrow K^m_{1,n}(\leftarrow J_{2,1}) \not\leftarrow K^0_{2,1} \leftarrow K^1_{2,1} \leftarrow \cdots$, with each $K^0_{u,1}$ becoming the commitment for the u-chain. Note that the commitment of an upper level chain is *implicitly* set as $J_{u,0} = K^m_{u-1,n}$.

In short, we are using salted functions to generate chains, and the lower level chain is a descendant of the next upper level chain. Index u may increase indefinitely while indices v and w are confined to n and m, respectively. See Fig. 1b for a diagram.

4.2.2 Communication Packets

For the framework of TinyOS, we design communication packets to fit within its 29-byte default payload size. It is trivial to allow larger packets if necessary. As depicted in Fig. 2, we define four types of packets that use the first byte of data payload for type distinction and the following four bytes for an index. Type 1 is an authenticated data packet of which 16 bytes are used for data transmission and the remaining 8 bytes are used for MAC generated by a lower level key. Type 2 is another form of authenticated data packet of which only 8 bytes are used for data transmission with an 8-byte MAC, while the remaining 8 bytes are used for key disclosure of a previous lower level interval. Type 3 is designed to handle sleeping nodes and idle sessions. It is the same with Type 2 except that the 8-byte data is a future lower level key masked with a future upper level key.

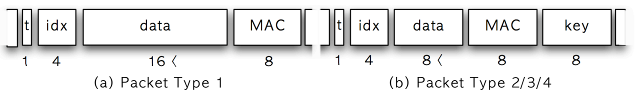


Fig. 2: Data payload in packets of X-TESLA.

The masked key is authenticated soon but unmasked much later. Of course, Type 2 and Type 3 can *trivially* be merged up to a single type of slightly larger packet. Type 4 packets hold a future lower level commitment at the data portion with a MAC calculated from an *upper* level key. Notice that the same lower level commitment is sent throughout a whole upper level chain. The AUX header field and the structure of CCM encryption mode of ZigBee [36] packets may be of some use in making more efficient variants of the packet types.

4.3 X-TESLA Details

4.3.1 Initialization

We assume a base station broadcasts authenticated messages to sensor nodes. A method to choose salt values is fixed at system design phase. The base station generates the first upper level chain by choosing seed key $J_{1,n} \in \mathcal{K}$ at random and also generates the first lower level chain together with the second upper level chain by choosing another seed key $J_{2,n} \in \mathcal{K}$ randomly. The values $J_{1,0} = F_1(J_{1,1}, S_{1,1})$ and $K^0_{1,1}$ are stored in each sensor node as initial upper and lower level commitments, respectively. Depending on the way salt is chosen, some extra information may also need to be stored. It would be advisable to keep these values secret until just before deployment. Generation of the second lower level chain together with the third upper level key chain should soon follow, so as to be ready for commitment distribution. When the initialized nodes are deployed, they are to be loosely time synchronized with the base station, as assumed in μTESLA.

4.3.2 Broadcast Authentication

During an $I^w_{u,v}$, the base station uses $K^w_{u,v}$ as the MAC key for Types 1, 2, and 3 packets being sent out, and reveals $K^w_{u,v}$ after a wait of time δ from the end of $I^w_{u,v}$, in Type 2 or 3 packets. We shall abuse interval indices, setting $I_{u,n+1} = I_{u+1,1}$, $I^{m+1}_{u,v} = I^1_{u,v+1}$, $I_{u,0} = I_{u-1,n}$, and $I^0_{u,v} = I^m_{u,v-1}$. The following is a Type 2 packet for use with "$\delta =$ *one* time interval." Here, | denotes concatenation and $*$ signifies the index and data portion.

$$T2P^w_{u,v} = \langle u, v, w\rangle \| \text{data} \| \operatorname{MAC}_{K^w_{u,v}}(*) \| K^{w-2}_{u,v}$$

This corresponds to what is usually stated as key disclosure delay of *two* time intervals. A sensor node may verify the MAC with $K^w_{u,v}$ if $F_1(K^w_{u,v}, S^w_{u,v}) = K^{w-1}_{u,v}$ is true, after some delay. Whenever a valid $K^w_{u,v}$ is received, it may replace the current commitment.

If the key disclosure message is lost, the sensor node buffers all messages it receives until a key disclosure message is successfully received, and computes $F^j_1(K^{w+j}_{u,v}, S^{w+\cdots}_{u,v})$ to obtain $K^w_{u,v}$, where F^j_1 is the j-times iterated application of F_1 (with appropriate salt values).

4.3.3 Commitment Hopping

With TESLA variants, there are at least two situations in which verification of a newly disclosed key places heavy computational load on a sensor node, resulting in many message drops, for the duration of this computation. First, if a sensor node falls into sleep mode or turns off its radio power to save energy, it may not be able to listen to the key disclosure messages during that period. Second, if there are long idle periods with no broadcast, it would be wasteful to disclose keys on schedule and a base station might minimize the key disclosures for those periods. As a result, there could be a large gap between the current commitment and the key to be verified. Type 3 packets can resolve this problem, by providing *commitment hopping*. Let $I_{u',v'}$ be an interval appearing after $I_{u,v}$. The distance[9] between the two intervals depends on the application needs. We set

$$T3P^w_{u,v} = \langle u, v, w\rangle \| K^m_{u',v'} \oplus J_{u',v'} \| \operatorname{MAC}_{K^w_{u,v}}(*) \| K^{w-2}_{u,v}.$$

The future lower level key $K^m_{u',v'}$ is masked by the future upper level key $J_{u',v'}$ and distributed in $I^w_{u,v}$. A node can authenticate it quickly within a few lower intervals, but unmask it afterwards only when $J_{u',v'}$ is obtained. After unmasking, it may replace the current lower level commitment, should it be older. In the opposite direction, $K^m_{u',v'}$ can be used to reveal $J_{u',v'}$. If v' is close to n, $K^m_{u',v'}$ can be used as the next upper level commitment.

4.3.4 Cross Authentication

With X-TESLA, keys of the upper level chain can be authenticated by the previous lower level chain since they are connected in a single chain by construction and since the latest commitment key of the previous lower level is available to sensor nodes. Type 3 packets further help in making this available. After any verification, the commitment for the upper level can be updated.

For authentication of a new lower level chain, the upper level chain is used. The following is a Type 4 packet. It distributes the commitment of the next lower level chain while disclosing a previous upper level key.

$$T4P_{u,v} = \langle u, v\rangle \| K^0_{u+1,1} \| \operatorname{MAC}_{J_{u,v}}(*) \| J_{u,v-1}$$

The next lower level commitment $K^0_{u+1,1}$ is distributed *at random instances* within $I_{u,v}$, authenticated with $J_{u,v}$. In fact, many (different) Type 4 packets are constructed and broadcast to deliver the same next lower level commitment $K^0_{u+1,1}$ during I_u. Therefore, a sensor node would have numerous chances to receive a correct $K^0_{u+1,1}$ during I_u, and resist DoS attacks without the huge buffers of multi-level μTESLA. A node buffers a single or slightly

9. The offset should be reasonably set. Multiple offsets may be used by assigning different message types to handle distinct offsets.

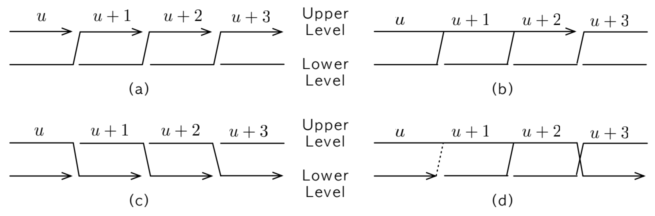


Fig. 3: Flexible constructions of X-TESLA. (a) Basic flow. (b) Extended flow. (c) Reversed flow. (d) Hybrid flow.

more[10] Type 4 packet appearing in the current interval $I_{u,v}$, and waits for a future Type 4 packet disclosing $J_{u,v}$ to verify it. If it is legitimate, the node takes the received $K^0_{u+1,1}$ to be the next lower level commitment and drops the data and MAC portions of all future Type 4 packets received during I_u. Otherwise, the sensor node replaces the buffered data with that of the new Type 4 packet hoping to verify it in a future interval. Thus, the required buffer size can be minimized. Even in the extreme case that a node fails to obtain correct commitments during I_u, the next upper chain authenticated by the current lower chain is still usable. More durable constructions are possible as discussed in Section 4.3.5.

As $J_{u,v-1}$ is disclosed by $T4P_{u,v}$, the first Type 4 packet of $I_{u,v}$ is broadcast after a delay of time $\gamma \geq \delta$ from the end of the interval $I_{u,v-1}$. The condition on γ is set because $J_{u,v-1}$ unmasks $K^m_{u,v-1}$, contained in a past Type 3 packet, and also because $J_{u,1}$ was used as a random seed for the previous lower level key chain via $K^m_{u-1,n} := F_1(J_{u,1}, S_{u,1})$. A Type 4 packet, in conjunction with a past Type 3 packet, allows for the commitment hopping, when u' used in $T3P^w_{u,v}$ is u or $u+1$.

4.3.5 Flexible Constructions

We now place more flexibility, in addition to the choice of chain lengths, into the X-TESLA construction. This will resolve even the most extreme situation that could occur with Type 4 packets. Starting from the basic flow of Fig. 3-(a), we extend the upper level chain over a number of lower level chains for better survivability against high communication faults and long idle sessions, as depicted in Fig. 3-(b). Even a short extension of the upper level chain with only small bits allows many lower level chains to be attached, and these may be generated on the fly. The extension increases stability of chain verification in both levels. The change also provides longer periods in which to distribute the next chain commitments for both levels through Type 3 and Type 4 packet variants.

The reverse flow depicted in Fig. 3-(c) allows reduction of Type 4 packets for environments in which authenticated messages are broadcast very frequently. Since an upper level chain serves as commitments for the next lower level chain, Type 4 packets distribute $J_{u+1,0} := F_0(J_{u+1,1}, S_{u+1,1})$ instead of $K^0_{u+1,1}$ in this version. But the dependance on Type 4 packets is smaller, because the upper level keys can be recovered stably from Type 3 packets if the authenticated broadcasts of the lower level are very frequent. The hybrid flow of Fig. 3-(d), offers *extreme durability*. For every $u \equiv 1 \pmod 3$, a lower chain of I_{u+3} is generated from a random seed with upper chains of I_{u+2}, I_{u+1}, and I_u, together with lower chains of I_{u+1} and I_{u-1} descending from this. The dotted line signifies that the starting lower chain of I_1 is an exception, having been generated from the upper level.

10. Within each $I_{u,v}$, a random selection process (which might come naturally from the environment) can also be employed. Unlike DoS tolerant μTESLA, even buffer sizes of 2 or 3 are extremely effective.

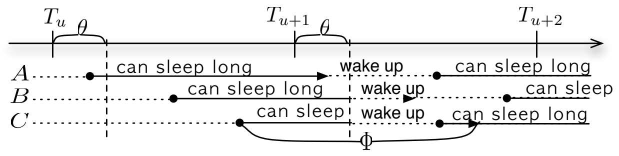


Fig. 4: Sleep mode in X-TESLA.

4.3.6 Sleep Mode Management

Energy efficiency is mandatory for sensor networks since tiny nodes are operated on batteries. Various types of sleep modes[11] that stop CPUs or radio functions are commonly used but care must be taken [16], as nodes that have been inactive for a long time may need to do much computation for key verification or lose commitment.

Let T_u denote the starting time of interval I_u, and set $\Phi = T_{u+1} - T_u$ to the length of one upper level chain. As depicted in Fig. 4, a sensor node shall not be allowed to go into a long term sleep or, at the least, not be allowed to stop radio functions for a long term period unless it has obtained the next lower level commitment, while short term sleeps are always allowed. More specifically, we fix some threshold value θ that takes the clock discrepancy of nodes into account, and for a node that has verified a Type 4 packet at time T, we allow it to set the maximum sleep length `timer()` to the duration of up to Φ only if $T < T_u + \theta$ (as in node A of Fig. 4), and to the duration of up to $T_{u+1} + \theta - T$ if otherwise (as in nodes B and C of Fig. 4). These values are meant to be the maximum sleeping length, and a sensor node may repeat going to sleep and waking up freely (or stopping and awaking radio components) within the given duration. The value θ should be fixed so that the security parameter $\varepsilon = P[\Phi] - P[\Phi - \theta]$ is kept appropriately small, where $P[\Phi]$ and $P[\Phi - \theta]$ denote the probabilities for a node to receive and verify a Type 4 packet within respective time lengths. Note that Φ is quite long, amounting to 3.6

11. Atmel's ATmega 128L CPU used in Berkeley Mica-Z provides six types of sleep modes named idle, ADC noise reduction, power-down, power-save, standby, and extended standby modes [6]. Among them, two standby modes reawake a mote in only six clock cycles (less than 1μs) and put it to sleep for a very short period. Thus it is quite unlikely that the node will sleep through a long term duration. The power-down mode stops the timer oscillator and all clocks, and the node eventually has to be reset for wake-up. As the power-save mode stops all components except for timer oscillator, timer/counter clock, and SRAM, and may last quite long, it should be most effective.

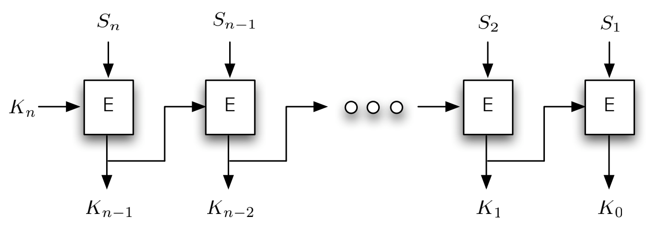


Fig. 5: Basic one-way chain using a blockcipher. (Salt does not require additional encryption for chains.)

hours if, for example, 2^{16}-key chains of 200ms intervals are used for the lower level. As depicted in Fig. 4, when a sensor node finally awakes, it should be allowed to go back to sleep for a long period only if it receives and verifies the next lower level commitment. If a node fails to verify any Type 4 packet in some I_u, it should be made to try harder in the next interval I_{u+1}, for example, by sleeping less, but often Type 3 packets could already have provided the lower level commitment of I_{u+1}. The sleep mode management system explained here should make the extendable property of X-TESLA work stably.

Still, the upper level length in X-TESLA needs to be chosen carefully, so that unexpected length of communication failure does not completely disrupt the system. Though this makes parameter selection challenging, the lengthening of the upper level is relatively cheap, and the use of flexible construction of Fig. 3 is also possible.

4.4 Implementation of X-TESLA

4.4.1 A Practical Construction

We use 2^8-key chains for the upper level and 2^{16}-key chains for the lower level with 200ms intervals but various other combinations are also possible. The broadcast module is implemented by connecting Tmote Sky to a PC, and the receiving module is ported into Mica-Z, with 17KB of program memory of which 9KB are occupied by system. We set 28-byte[12] payloads. As we designed in Fig. 5, we utilize the 64-bit key version of RC5 for generating chains. The salt values, used as plaintext, should be known to the verifying node as well, and can be defined in various ways. Taking a practical approach, we use $S_{u,v} = \langle u, v\rangle$ and $S^w_{u,v} = \langle u, v, w\rangle$ in the implementation, where the indices are zero-extended to fill the 64-bit block size, with the exception of the most significant bit, which is used to differentiate F_0 from F_1.

We caution that this is *not a complete solution* against TMD-tradeoff attacks. If the indices are short, so that index repetition is common, an attacker may decide to focus on (multiple) target points corresponding to one fixed index. Even if index is long enough not to repeat itself within the lifetime of the network, a tradeoff attack on a single target would still be possible. This is not an immediate threat with average-powered attackers, but probably not so for long with 64-bit chains. Rather we propose a more robust solution to combine old (disclosed) key with the index to produce salt. For example, we set $S_{u,v} = \langle u, v\rangle \oplus J_{u-1,0}$ and $S^w_{u,v} = \langle u, v, w\rangle \oplus K^m_{u-2,0}$, where nodes keep $J_{u-1,0} (= K^m_{u-2,0})$ until finishing $I_{u,v}$ and the initial salt is specifically given. Thus, a sort of randomness, unpredictable until near the time of use, could be employed, so as to prevent pre-computation.

4.4.2 Running X-TESLA

To start with, we need a 2^8-key chain for the upper level and a $(2^{16} + 2^8)$-key chain for the lower level with its source, which is the next upper level. For commitments soon to be distributed, one additional future chain must be prepared. As a result, the base station maintains three upper level and two lower level chains at run time. It takes only 797ms to compute these chains on a PC with dual AMD Opteron 244 (1.8GHz) CPU and only 1MB to store them. In our test implementation, for simplicity, we preset the starting time and had the base station send out a synchronization command. This is acceptable, as the initial deployment phase is usually assumed to be secure in the literature. More sophisticated synchronization methods can be found from [9], [17] and [33].

The number of unauthenticated packets buffered by a sensor node depends on the period and reliability of key disclosure messages. Concerning the key disclosure interval, note that a 36-byte TinyOS packet, consisting of 5-byte header, 29-byte data payload, and 2-byte CRC tailer, takes 28.8ms to send on a 10kbps radio network, with round-trip taking less than 60ms. Similarly, a 39-byte ZigBee packet in which 29 bytes are data payload, takes 5.1ms on average to send on a 60kbps[13] radio network, with round-trip taking less than 15ms. So any key disclosure interval larger than 50ms is possible. In case the shorter 50ms intervals are used, it might be preferable to use slightly longer chains, to preserve the duration covered by a single lower level chain.

5 Analysis of X-TESLA

5.1 Security

5.1.1 General Issues

As was stated in Section 4.2.1, X-TESLA protects against TMD-tradeoff attacks through the explicit use of salt, so that even the 64-bit key chains can be practically secure. The extendable management of short chains leads to security advantages as well as efficiency advantages. In (multi-level) μTESLA, the lifetime of the sensor network is pre-determined and a chain (or at least one chain) that spans throughout this very long period is used. This means that the seed key (and far future keys) should be protected very securely, for were it to be compromised without the base station being aware, it could

12. Since the Tmote Sky has a 16-bit CPU manipulating 2-byte words, we use 28-byte payloads, so as not to exceed the 29-byte default setting of TinyOS and borrow 2-bits from the index field for type definition.

13. We have checked that the transmission rate of a single Mica-Z (with a CC2420 chip) is slightly greater than 60kbps for 39-byte packets and 120kbps for the maximum size 126-byte packets, while the maximum for ZigBee radio is around 250kbps.

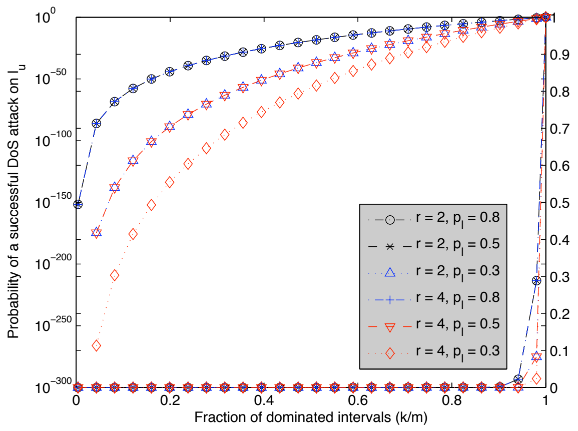


Fig. 6: The success probability of a DoS attack on I_u, utilizing k-interval domination per each $I_{u,v}$. (Parameters $m = 2^8$ and $n = 2^6$ are assumed with r Type 4 packets broadcast in each $I_{u,v}$ with p_l packet losses rate. Two y-axes are used for detailed visuality: the left one in a logarithmic scale and the right one in a normal scale.)

be troublesome for a very long period. So, depending on the adversary model, X-TESLA, which uses short-lived chains, will have security advantages. In any case, using long chains is less than ideal, as function iteration continually reduces the entropy of key space.

In X-TESLA, time is divided into minute intervals with $K^w_{u,v}$ set as the authentication key for the $\langle u, v, w\rangle$-th lower level interval $I^w_{u,v}$ starting at time $T^w_{u,v}$, and also into larger intervals with $J_{u,v}$ set as the key for the $\langle u, v\rangle$-th upper level interval $I_{u,v}$ starting at time $T_{u,v}$. Packets authenticated with a certain correct key, but received after the key was disclosed, taking the possible clock discrepancy into account, should be dropped by a receiver. Explicitly, for a message authenticated with $J_{u,v}$ and received at time T, its acceptance would be based on whether the security condition $T - T_{u,v+1} + \epsilon < \gamma$ is satisfied[14], where ϵ is the allowed clock discrepancy between the sender and receivers. Similarly, for a message authenticated with $K^w_{u,v}$ and received at time T, we would check $T - T^{w+1}_{u,v} + \epsilon < \delta$. This stops a bogus message injected by an adversary from being authenticated.

5.1.2 DoS Attack Resistance

Communication faults and DoS attacks may result in *packet loss* or *forged packets*. To overcome these problems, a base station could repeat a packet for a reasonable number of times. For example, if a packet loss rate is 30%, the probability of receiving can be increased to 99.2% by repeating the packet just four times. Since the time interval of lower level chains is tiny and the verification key is disclosed shortly, forged messages can be deterred by an affordable buffering in the lower level.

Compared to the other types, Type 4 packets could be less resistant to DoS attacks because they have to be buffered until verification for the duration of the longer upper interval. By jamming a whole interval[15] $I_{u,v}$, a DoS attacker can drop all Type 4 packets from that interval, but fortunately the impact diminishes rapidly as the attacker loses domination, especially when considered over all of I_u. Let p_l be the packet loss rate of sensor nodes due to communication faults and sleep modes and set $\bar{p}_l = 1 - p_l$. Suppose that the base station randomly chooses r of the m intervals within each $I_{u,v}$ to broadcast Type 4 packets and that the attacker dominates k intervals within each $I_{u,v}$. The probability p_d that a DoS attacker is successful in I_u can be calculated as

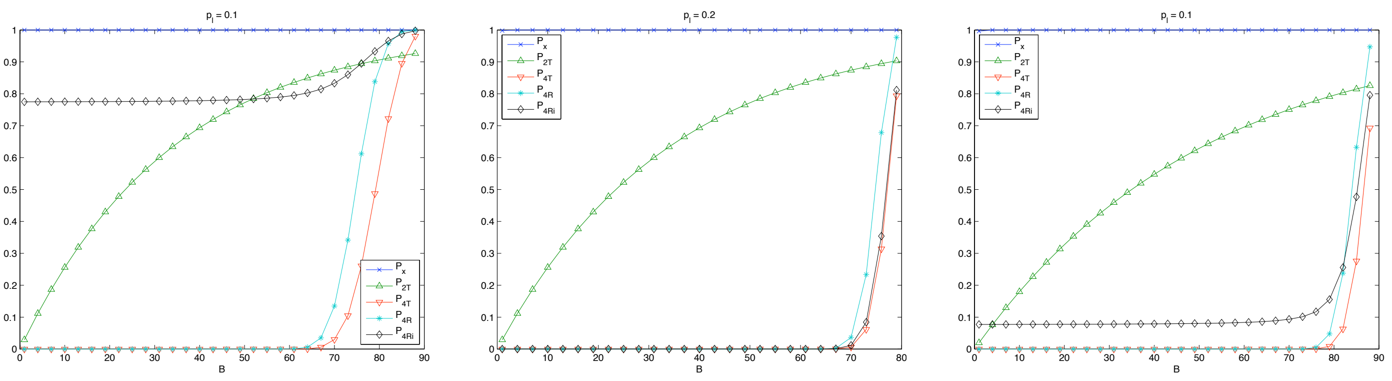
$$p_d = \left[\binom{k}{\lceil \bar{p}_l r \rceil} \Big/ \binom{m}{\lceil \bar{p}_l r \rceil} \right]^{n-1},$$

where the numerator is set to be zero when $k < \lceil \bar{p}_l r \rceil$. Fig. 6 depicts it and shows p_d to be extremely low unless the attacker dominates most of the nm channels. The curves leaning on the right y-axis shows how large k is necessary in each $I_{u,v}$ to increase p_d, while the curves filling the top left corner represent it in a logarithmic scale. When 2^6-key chains and 2^{14}-key chains are employed for respective levels with 200ms lower level interval, and two Type 4 packets are distributed in each $I_{u,v}$ over ZigBee channels for a 0.9 hour period, which amounts to only 0.000095% communication overhead, we can see that p_d is lower than 0.0015 for very high packet loss rate of $p_l = 0.8$ until the attacker has control over 90% of each $I_{u,v}$ interval and it grows only to 0.022 for 95%. When 2^8-key upper chains and 2^{16}-key lower chains are considered, the DoS success probability p_d drops to 0.0003 even when all but 3% of the intervals is dominated. Thus the more durable constructions of Fig. 3 that use longer upper chains strengthen our scheme further.

Since the time interval of upper chains is relatively long, the attacker could try to overflow sensor node buffers with forged Type 4 packets after listening to the correct key $J_{u,v-1}$ disclosed in that interval. However, it is sufficient with X-TESLA that each node buffers only a single (or slightly more) Type 4 packet received in each interval $I_{u,v}$ for verifying $K^0_{u+1,1}$ within I_u. Among the four constructions of X-TESLA (Fig. 3), the basic method delivers the next lower level chain commitment through Type 4 packets, but repeats it within I_u, so that a node can receive a valid one with very high probability. The other three constructions allow even better probability. Consequently, X-TESLA resists DoS attacks of forged packets intrinsically, whereas multi-level μTESLA necessitates a large buffer and much precomputation with storage for its CDM packets. We could observe at least such a big difference between them.

14. For intervals whose upper level key was used in Type 3 masking, δ should be used in place of γ.

15. The same Type 4 packet may be repeated at random instances within an $I_{u,v}$, while the same key is authenticated in distinct Type 4 packets during an I_u. From the perspective of a DoS attacker, each Type 4 packet of $I_{u,v}$ shall be distributed at randomly chosen lower level interval $I^w_{u,v}$ by the base station.



(a) (b) (c)

Fig. 7: Probability of proper Type 4/CDM handling with a B-size buffer. (B: 24-byte unit for DoS resistant μTESLA and 16-byte units for the others.) (a) $f = 100, r = 3, p_l = 0.1$. (b) $f = 100, r = 3, p_l = 0.2$. (c) $f = 100, r = 2, p_l = 0.1$.

5.2 Efficiency Comparison

While multi-level μTESLA and X-TESLA provide comparable resistance against DoS attacks, in this section, we show that the required resources are different.

5.2.1 Computation and Storage for Base Stations

With X-TESLA, only a small number of short chains need to be stored in the base station, with the rest computed on the fly, and the chains are extendable indefinitely. In μTESLA and DoS resistant multi-level μTESLA, full chains covering all of the expected lifetime (and their CDMs in multi-level μTESLA) have to be pre-computed and stored. By storing the pre-computed chain only in part, storage can be reduced, but at the cost of online *re*computation. Details are discussed in Appendix E.

5.2.2 Storage for Sensor Nodes

To verify a message, a sensor node has to buffer the index, data, and MAC fields until the delayed exposure of the corresponding key. This is a shared property of all μTESLA variants. X-TESLA shares another property with multi-level μTESLA in that new commitments for future chains need to be buffered and verified, but X-TESLA requires less storage in the nodes than multi-level μTESLA for three reasons. First, only two levels are used in X-TESLA, while more levels (or longer chains) are necessary in multi-level μTESLA. Second, X-TESLA verifies an upper level commitment, which is masked for later use, almost immediately after reception, following the shorter lower level schedule, but with multi-level μTESLA, verification of an level-ℓ commitment must wait through level-$(\ell{+}1)$'s longer interval, and this situation worsens as we go up the levels. Third, verification of lower level commitment in X-TESLA follows the upper level interval schedule, but without large buffering.

With X-TESLA, a sensor node stores one most recently authenticated key as the current commitment for each level, along with the next lower level commitment, and possibly the masked key from a recent Type 3 packet, adding up to a total of four keys (taking 32 bytes) at runtime. In comparison, an M-level μTESLA node stores $3M-2$ keys (taking *more* bytes), along with the buffered CDMs for each level except the highest level. Let us now look at the node storage required to handle Type 4 or CDM packets reliably, by comparing an X-TESLA of $2^8/2^{16}$-key upper/lower chains with a 2-level μTESLA of $2^{12}/2^{16}$-key chains and a 4-level of $2^4/2^8/2^8/2^8$-key chains within the 2^{28}-key lifetime. Let r and f denote the number of real and forged Type 4/CDM packets appearing in a single $I_{u,v}$ or lowest level 2^8-key interval. To be fair, these are set to the same value for all schemes. Assuming a node buffer of size B assigned to the processing of Type 4/CDM packets, the probability that a node fails to buffer any real packets within a 2^8-key interval is

$$\rho = \binom{f'}{B} \Big/ \binom{f'+r'}{B},$$

where $f' = \bar{p}_l f$ and $r' = \bar{p}_l r$. Then, the success probability of an X-TESLA node obtaining a proper Type 4 packet during a 2^{16}-key period can be written as

$$P_X = 1 - \rho^{2^8},$$

and the same can be written as

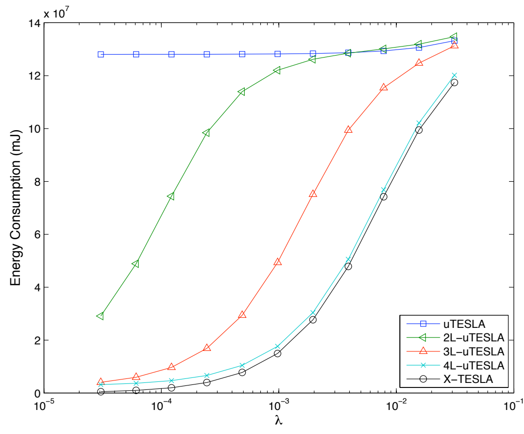
$$P_{2T} = 1 - \binom{2^8 f'}{B} \Big/ \binom{2^8(f'+r')}{B},$$

for a 2-level DoS tolerant μTESLA. A parallel figure for a 4-level DoS tolerant μTESLA would be

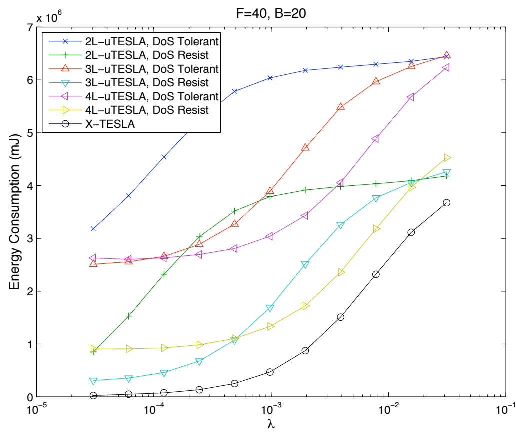
$$P_{4T} = (1-\rho)^{2^8}$$

the probability of obtaining a CDM from each 2^8-key interval. This is depicted in Fig. 7 and it can be seen that X-TESLA performs exceptionally well regardless of buffer size, whereas large buffers are necessary to achieve reasonable probability in the other types.

Let us also consider the DoS resistant version of multi-level μTESLA. In this case, assuming possession of the current CDM, the probability $1 - p_l^r$ of obtaining CDM for the next interval is quite large. But the probability of consecutive proper receptions decreases very quickly,



(a)



(b)

Fig. 8: Energy consumption of a sensor node with regard to computation and communication. (In Mica-Z, ATmega128 CPU needs 33mW active power and 75μW sleep power, and the CC2420 radio needs 38mW receive power [30]. In our experiment, it took 14.5ms (0.477mJ) to compute a single key following the basic chain, 15.4ms (0.508mJ) to compute CBC-MAC of two input blocks, and 16.3ms (0.539mJ) for CBC-MAC of three blocks, using RC5. It cost 0.106mJ to receive a 39-byte ZigBee packet and 0.114mJ for a 47-byte packet.) (a) Without forged CDMs; $\mathcal{F} = 0$. (b) With forged CDMs; $\mathcal{F} = 40$, buffer $B = 20$ for multi-level μTESLA and $B = 1$ for X-TESLA.

and when a CMD is lost, one must fall back to the DoS tolerant mode. To maintain high CDM reception probability even in this situation both 2-level and 4-level DoS resistant μTESLA need to be equipped with buffer size comparable to those of DoS tolerant versions. To make a more direct comparison, let us relax our condition and compute the probability of a DoS resistant 4-level μTESLA obtaining CDMs from all intervals, with the exception of one. If immediate authentication of CDMs is conducted without buffering, this is given by

$$P_{4Ri} = (1 - p_l^r)^{2^8} + p_l^r \sum_{i=0}^{2^8-1} (1 - p_l^r)^i (1 - \rho)^{2^8-i-1},$$

and, when otherwise, by

$$P_{4R} = (1 - \rho)^{2^8} + \rho \sum_{i=0}^{2^8-1} (1 - \rho)^i (1 - \rho)^{2^8-i-1}.$$

These are all depicted in Fig. 7, where the exceptional performance of X-TESLA is evident. Comparison of buffer needed for fair performance under nice conditions (Fig 7a) and when either p_l is slightly larger (Fig 7b) or r/f is slightly smaller (Fig 7c) shows that buffer size is critical for μTESLA variants other than X-TESLA.

Note that the above estimations are valid only when r' is not too small, as they assume at least one proper Type 4 or CDM packet arriving at the sensor nodes. This interesting exception is treated in Appendix F.

5.2.3 Computation and Communication for Sensor Nodes

In sensor networks, power consumption of sensor nodes is one of the most significant issues since sensor nodes are usually operated on batteries. With μTESLA variants, sensor nodes may consume energy while computing chains (for verification), computing MAC, and receiving broadcast packets. We analyze computation and communication costs of sensor nodes from this perspective.

Let a random process $X(t)$ have an exponential distribution and let $E[X]$ be the expected value that is the average distance between two packet arrivals, with regard to a message rate. We take a *slot* to be the time interval for which a single lowest level key is valid. For example, with X-TESLA above, one upper level interval covering 2^8-key lower level chain is said to have 2^8 slots. Let α be the number of slots within an upper level interval, and let $\mathcal{T}$ be the total number of slots within the whole duration of the key chains, or usage lifetime. Then we define

$$C_{i,\alpha} = \begin{cases} E[X] & \text{when } \sigma_i \bmod \alpha \geq E[X], \\ \sigma_i \bmod \alpha & \text{otherwise,} \end{cases}$$

the number of slots requiring computation for verification of the i-th packet that has just arrived, where, $\sigma_i = i \cdot E[X]$. We assume that r legitimate CDMs and $\mathcal{F}$ forged CMDs are received in each interval for simplicity. Packet losses are also disregarded for the same reason but can be added trivially. Using these notions, we can write the energy consumption $\mathcal{E}$ of each μTESLA variant as the sum of costs from the following operations: (1) key computation whose number is given in terms of $C_{i,\alpha}$, (2) MAC computation and message radio reception, both of which counts to $\left\lfloor \frac{\mathcal{T}}{E[X]} \right\rfloor$, and (3) buffering and verifying of both real and adversarial CDM or Type 3/Type 4 packets. Complete equations are described in Appendix G. By substituting explicit values obtained through experiments into these equations, we arrive at the comparisons of Fig. 8. In terms of energy consumption, as can be seen in Fig. 8-(a), both X-TESLA and 4-level μTESLA are superior to others unless there are forged packets (coming from DoS attacks). If there are forged packets, X-TESLA consumes less energy than 4-level μTESLA, as in Fig. 8-(b). With X-TESLA, a sensor node can skip unnecessary key computation and commitment verification to save energy.

6 Conclusion

Through the application of TMD-tradeoff techniques we observed that care should be taken with the short-key chain based broadcast authentication schemes. We have proposed X-TESLA, an efficient scheme which may continue indefinitely and securely, that addresses this and many other issues of the previous schemes. With the advent of more powerful sensor node commodities such as iMote2 [14], the future of public-key technique application to broadcast authentication looks bright, but X-TESLA can efficiently be combined with public-key techniques also. For example, we could modify X-TESLA to use digital signatures on Type 4 packets, keeping everything else the same.

Acknowledgment

Authors thank deeply Virgil Gligor, Adrian Perrig, Jung Hee Cheon, JongHyup Lee, and anonymous reviewers for their most helpful comments to improve this paper.

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A Survey on Sensor Networks," IEEE Comm. Magazine, Vol. 40, No. 8, pp. 102–114, Aug. 2002.

[2] G. Avoine, P. Junod, and P. Oechslin, "Time-Memory Trade-offs: False Alarm Detection Using Checkpoints," Indocrypt 2005, LNCS 3797, pp. 183–196, Springer-Verlag, 2005.

[3] S. Babbage, "Improved Exhaustive Search Attacks on Stream Ciphers," European Convention on Security and Detection, IEE Conference Publication No. 408, pp. 161–166, IEE, 1995.

[4] A. Biryukov and A. Shamir, "Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers," Asiacrypt 2000, LNCS 1976, pp. 1–13, Springer-Verlag, 2000.

[5] B. Calhoun, D. Daly, N. Verma, D. Finchelstein, D. Wentzloff, A. Wang, S. Cho, and A. Chandrakasan, "Design Considerations for Ultra-Low Energy Wireless Microsensor Nodes," IEEE Trans. Computers, vol. 54, no. 6, pp. 727–740, Jun. 2005.

[6] Crossbow Technology, Inc. `http://www.xbow.com`.

[7] A. Durresi, V. Paruchuri, S. Iyengar, and R. Kannan, "Optimized Broadcast Protocol for Sensor Networks," IEEE Trans. Computers, vol. 54, no. 8, pp. 1013–1024, Aug. 2005.

[8] P. Flajolet and A. M. Odlyzko, "Random Mapping Statistics," Eurocrypt 1989, LNCS 434, pp. 329–354, Springer-Verlag, 1990.

[9] S. Ganeriwal, S. Capkun, C. Han, and M. Srivastava, "Secure Time Synchronization Service for Sensor Networks," Proc. ACM Workshop on Wireless Security (WiSe), pp. 97–106, 2005.

[10] J. Dj. Golić, "Cryptanalysis of Alleged A5 Stream Cipher," Eurocrypt 1997, LNCS 1233, pp. 239–255, Springer-Verlag, 1997.

[11] M. Hellman, "A Cryptanalytic Time-Memory Trade-off," IEEE Trans. on Infor. Theory, 26, pp. 401–406, 1980.

[12] J. Hong and P. Sarkar, "New Applications of Time Memory Data Tradeoffs," Asiacrypt 2005, pp. 353–372, Springer-Verlag, 2005.

[13] Y. Hu, M. Jakobson, and A. Perrig, "Efficient Constructions for One-way Hash Chains," Proc. ACNS 05, LNCS 3531, pp. 423–441, Springer-Verlag,2005

[14] Intel IMote2 Overview, `http://www.intel.com/research/downloads/imote_overview.pdf`, 2005. Commercialized by Crossbow, INC. `http://www.xbow.com/`.

[15] K. Kar, A. Krishnamurthy, and N. Jaggi, "Dynamic Node Activation in Networks of Rechargeable Sensors," IEEE/ACM Trans. Networking, Vol. 14, No. 1, pp. 15–25, Feb. 2006.

[16] M. Karaata and M. Gouda, "A Stabilizing Deactivation/Reactivation Protocol," IEEE Trans. Computers, vol. 56, no. 7, pp. 881–888, Jul. 2007.

[17] Q. Li and D. Rus, "Global Clock Synchronization in Sensor Networks," IEEE Trans. Computers, vol. 55, no. 2, pp. 214–226, Feb. 2006.

[18] D. Liu, P. Ning, "Efficient Distribution of Key Chain Commitments for Broadcast Authentication in Distributed Sensor Networks," Proc. ISOC Network and Distributed System Security Symposium (NDSS), pp. 263–276, Feb. 2003.

[19] D. Liu and P. Ning, "Multi-Level μTESLA: Broadcast Authentication for Distributed Sensor Networks," ACM Trans. Embedded Computing Systems, Vol. 3, No. 4, pp. 800–836, Nov. 2004.

[20] M. Luk, A. Perrig, and B. Willock, "Seven Cardinal Properties of Sensor Network Broadcast Authentication," Proc. ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN), October 2006.

[21] M. Luk, G. Mezzour, A. Perrig, and V. Gligor, "MiniSec: A Secure Sensor Network Communication Architecture," Proc. ACM/IEEE Conf. Information Processing in Sensor Networks (IPSN), April 2007.

[22] D. J. Malan, M. Welsh, and M. D. Smith, "A Public-key Infrastructure for Key Distribution in TinyOS Based on Elliptic Curve Cryptography," Proc. IEEE International Conf. on Sensor and Ad Hoc Comm. and Network, Oct. 2004.

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

[24] Oechslin, "Making a Faster Cryptanalytic Time-Memory Trade-off," Crypto 2003, LNCS 2729, pp. 617–630, Springer-Verlag, 2003.

[25] J. Park and S. Sahni, "Maximum Lifetime Broadcasting in Wireless Networks," IEEE Trans. Computers, vol. 54, no. 9, pp. 1081–1090, Sep. 2005.

[26] A. Perrig, R. Canetti, D. Song, and D. Tygar, "Efficient Authentication and Signing of Multicast Streams over Lossy Channels," Proc. IEEE Security and Privacy Symposium, May 2000.

[27] A. Perrig, R. Canetti, D. Song, and D. Tygar, "Efficient and Secure Source Authentication for Multicast," Proc. ISOC Network and Distributed System Security Symposium (NDSS), Feb. 2001.

[28] A. Perrig, R. Szewczyk, V. Wen, D. Cullar, and J. D. Tygar, "SPINS: Security Protocols for Sensor Networks," Proc. ACM/IEEE International Conf. on Mobile Computing and Networking, pp. 189–199, July 2001.

[29] A. Perrig, J. Stankovic, and D. Wagner, "Security in Wireless Sensor Networks," Comm. ACM, vol. 47, no. 6, pp. 53–57, June 2004.

[30] J. Polastre, R. Szewczyk and D. Culler, "Telos: Enabling Ultra-Low Power Wireless Research," Proc. International Conf. on Information Processing in Sensor Networks, 2005.

[31] R. Rivest, "The RC5 Encryption Algorithm," FSE 1994, LNCS 1008, pages 86–96, Springer-Verlag, 1995.

[32] P. Rogaway, M. Bellare, and J. Black, "OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption," ACM TISSEC, Nov. 2001.

[33] K. Sun, P. Ning, C. Wang, A. Liu, and Y. Zhou, "TinySeRSync: Secure and Resilient Time Synchronization in Wireless Sensor Networks," Proc. ACM Conf. on Comp. and Comm. Security (CCS), 2006.

[34] W. Ye, J. Heidemann, and D. Estrin, "Medium Access Control with Coordinated Adaptive Sleeping for Wireless Sensor Networks," IEEE/ACM Trans. Networking, vol. 12, no. 3, pp. 493–506, June 2004.

[35] S. Zhu, S. Setia, and S. Jajodia, "LEAP: Efficient Security Mechanisms for Large-scale Distributed Sensor Networks." Proc. ACM Conf. on Comp. and Comm. Security (CCS), pp. 62–72, ACM Press, 2003.

[36] ZigBee Specification Ver. 1.0, `http://www.zigbee.org`, 2005.

APPENDIX A
ATTACK TARGET ENVIRONMENT ADVOCATION

The target system of TMD-tradeoff attack presented in this paper was a 64-bit key chain system with a 200ms disclosure interval. Let us explain that this is (or was) a practical choice of parameters for sensor networks.

The main objection would be that the 64-bit key size is too small. Most cryptographic systems today use 128-bit keys, or at least 80-bit keys. But in the sensor network world, communication and storage is extremely costly and the size of communication packets is limited. Consequently, shorter keys are required. For example, the paper [19] gives a performance evaluation of their multi-level μTESLA protocol, using 8-byte keys. In the LEAP paper [35], 8-byte keys are mentioned in the course of discussing storage requirements. Finally, the SPINS paper [28] mentions the use of RC5 for chain creation, which is normally taken to be a 64-bit blockcipher. So there is enough evidence that 64-bit keys are considered appropriate in the sensor network environment. In practice, both TinyOS and ZigBee standard within the TinyOS framework use 29-byte payloads in their 36-byte and 45-byte packets, respectively, showing how restrictive the sensor network environment is.

While the use of 64-bit keys may seem strange to cryptographers, this has been considered acceptable because of the way it is used. The key needs to remain secret for only a very short period. For example, in our setting, the attacker would need to recover the 64-bit key from a MAC value in much less than 200ms for the recovered key to be of any use. This is infeasible with a straightforward approach for the moment.

Concerning the key disclosure interval, we have taken the reasonable 200ms value as an example, but as was explained in Section 3.4.2, with a full understanding of our analysis it becomes clear that no small tweaking of this parameter makes the system immune to TMD-tradeoff attacks.

Another line of objection dismisses tradeoff attacks, saying that the use of key indices as auxiliary input to chain computation renders any pre-computation attacks ineffective. Indeed, our protocol X-TESLA follows this line of thought by incorporating salt values. Well-formed indices could serve as salt, but as there is a practical limit to the index length, this cannot be a complete solution. Some other unpredictability, such as part of previously disclosed keys have to be employed to be more robust. In any case, we were unable to find any mentioning of one-way chain salting in the μTESLA related literature. This is not too surprising, because straightforward application of tradeoff attack to the obvious one-way function, namely, the one used for chain creation, does not bring about anything useful to the attacker. At least two tricks were involved in our application of the tradeoff technique.

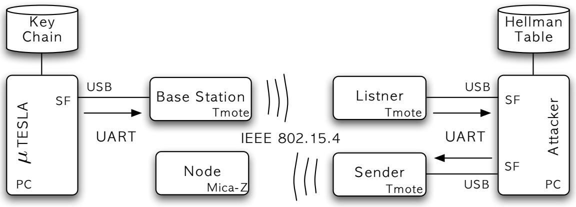


Fig. 9: Our Attack Experiment of μTESLA.

APPENDIX B
EXPERIMENT DETAIL

As was explained in Section 3.3, we did multiple tests with independent sets of target data, but utilizing the same Hellman table. From each of the tests, multiple inverses were found, most of which were *useless* inverses, as explained in Section C. Of the ten independent tests we did, eight returned (sometimes multiple) correct inverses, so we had an 80% attack success rate. Relevant counts are given in Table 1. The table also lists number of *false alarms* obtained during each test. These amount to less than 0.4% of $2^{24.04}$, the total number of target points we used, and caused only a negligible amount of added online time in our tests.

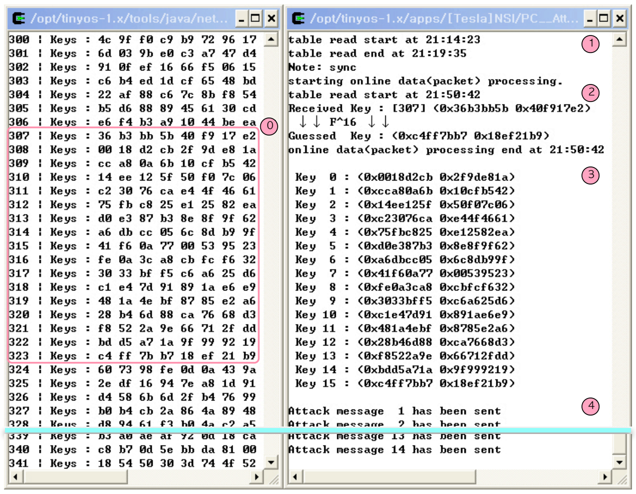
As was explained in Section 3.3.4, we also conducted the experiment of utilizing the acquired correct future keys, to check if the online phase of our attack could really be done in real time, in our test-bed environment using real sensor nodes. Fig. 9 depicts the attack experiment model.

The experiment result is depicted in Fig. 10. It takes the attack program 5.2 minutes to load Hellman table[16] onto its memory (part ①). It then listens on the radio channel through the Listener and starts the online data processing. We can see the base station disclosing key `36b...e2` at point ⓪. It took 31.1 minutes to reach this point. At almost the same time, this disclosed key is found by the attack program to be covered by the Hellman table, and its 16-step pre-image key `c4f...b9`

16. The Cygwin Unix environment we used came with a 1.5GB memory bound, so only 1.3GB of the 1.5GB Hellman table was loaded.

	total inverse	correct inverses	false alarms
test0	19	0	67151
test1	21	1	67609
test2	21	1	67850
test3	19	1	68075
test4	11	0	67655
test5	15	3	67921
test6	24	3	67291
test7	19	1	67619
test8	14	1	67816
test9	19	1	67252
average	18.2	-	67623.9

TABLE 1: Various counts for each test


/opt/tinyos-1.x/tools/java/net...
/opt/tinyos-1.x/apps/[Tesla]NSI/PC__Att...
0
1
2
3
4

```
300 ¦ Keys : 4c 9f f0 c9 b9 72 96 17
301 ¦ Keys : 6d 03 9b e0 c3 a7 47 d4
302 ¦ Keys : 91 0f ef 16 66 f5 06 15
303 ¦ Keys : c6 b4 ed 1d cf 65 48 bd
304 ¦ Keys : 22 af 88 c6 7c 8b f8 54
305 ¦ Keys : b5 d6 88 89 45 61 30 cd
306 ¦ Keys : e6 f4 b3 a9 10 44 be ea
307 ¦ Keys : 36 b3 bb 5b 40 f9 17 e2
308 ¦ Keys : 00 18 d2 cb 2f 9d e8 1a
309 ¦ Keys : cc a8 0a 6b 10 cf b5 42
310 ¦ Keys : 14 ee 12 5f 50 f0 7c 06
311 ¦ Keys : c2 30 76 ca e4 4f 46 61
312 ¦ Keys : 75 fb c8 25 e1 25 82 ea
313 ¦ Keys : d0 e3 87 b3 8e 8f 9f 62
314 ¦ Keys : a6 db cc 05 6c 8d b9 9f
315 ¦ Keys : 41 f6 0a 77 00 53 95 23
316 ¦ Keys : fe 0a 3c a8 cb fc f6 32
317 ¦ Keys : 30 33 bf f5 c6 a6 25 d6
318 ¦ Keys : c1 e4 7d 91 89 1a e6 e9
319 ¦ Keys : 48 1a 4e bf 87 85 e2 a6
320 ¦ Keys : 28 b4 6d 88 ca 76 68 d3
321 ¦ Keys : f8 52 2a 9e 66 71 2f dd
322 ¦ Keys : bd d5 a7 1a 9f 99 92 19
323 ¦ Keys : c4 ff 7b b7 18 ef 21 b9
324 ¦ Keys : 60 73 98 fe 0d 0a 43 9a
325 ¦ Keys : 2e df 16 94 7e a8 1d 91
326 ¦ Keys : d4 58 6b 6d 2f b4 76 99
327 ¦ Keys : b0 b4 cb 2a 86 4a 89 48
328 ¦ Keys : d8 94 61 f3 b0 4a c2 a5
339 ¦ Keys : b3 a0 ae af 92 0d 18 ca
340 ¦ Keys : c8 b7 0d 5e bb da 81 00
341 ¦ Keys : 18 54 50 30 3d 74 4f 52
```

```
table read start at 21:14:23
table read end at 21:19:35
Note: sync
starting online data(packet) processing.
table read start at 21:50:42
Received Key : [307] (0x36b3bb5b 0x40f917e2)
 ↓↓ F^16 ↓↓
Guessed  Key : (0xc4ff7bb7 0x18ef21b9)
online data(packet) processing end at 21:50:42

 Key  0 : (0x0018d2cb 0x2f9de81a)
 Key  1 : (0xcca80a6b 0x10cfb542)
 Key  2 : (0x14ee125f 0x50f07c06)
 Key  3 : (0xc23076ca 0xe44f4661)
 Key  4 : (0x75fbc825 0xe12582ea)
 Key  5 : (0xd0e387b3 0x8e8f9f62)
 Key  6 : (0xa6dbcc05 0x6c8db99f)
 Key  7 : (0x41f60a77 0x00539523)
 Key  8 : (0xfe0a3ca8 0xcbfcf632)
 Key  9 : (0x3033bff5 0xc6a625d6)
 Key 10 : (0xc1e47d91 0x891ae6e9)
 Key 11 : (0x481a4ebf 0x8785e2a6)
 Key 12 : (0x28b46d88 0xca7668d3)
 Key 13 : (0xf8522a9e 0x66712fdd)
 Key 14 : (0xbdd5a71a 0x9f999219)
 Key 15 : (0xc4ff7bb7 0x18ef21b9)

Attack message  1 has been sent
Attack message  2 has been sent
Attack message 13 has been sent
Attack message 14 has been sent
```

Fig. 10: Result of attack on μTESLA with 64-bit key chain

is printed (step ②). The attack program suspends its online data processing and computes 16 keys (step ③) by iteratively applying F, starting from the guessed key `c4f...b9`. Key 0 and Key 1 are discarded, as these are already invalid due to the two-interval delayed disclosure schedule. Finally, the attack program sends out forged messages, one in each interval, using the remaining 14 keys (step ④). We could visually check the red LED flashing as commanded through the forged messages. Note that more messages can be sent out in each interval, for example, 39 packets in each interval and as many as 551 packets in the 14 intervals, on the 60kbps radio channel of our experiment.

APPENDIX C
TECHNICAL COMPLICATIONS OF ATTACK ALGORITHM

There are some technical detail that need to be discussed for a full understanding of our attack algorithm.

C.1 Useless Inverses

The values x returned by Algorithm 2 are guaranteed to satisfy $H(x) = F\circ\cdots\circ F(x) = y$ for some given disclosed key $y \in \hat{\mathcal{D}}$, but due to the fact that F is not injective, this does not necessarily imply that x is the correct key we are looking for.

Let us explain with an example. One of the answers returned by the online phase (Algorithm 2) of our first test was

`0xdb52a3dfd4257c4f maps to`
`0x41fda1a6e7439711 under` H.

This implies, in particular, that the target point `0x41f`$\cdots$`11` appears in the target data set $\hat{\mathcal{D}}$. To be more precise, it was the 316362-th key disclosed after the original commitment. Since $316362 = 17\cdot(5\cdot 60\cdot 60)+10362$, in the 18-th (lowest level) chain of the first day, we have

$$K_{10362} = \texttt{0x41fda1a6e7439711}.$$

Below, we have listed 16 more keys that would be released after K_{10362}.

	original target chain	chain constructed from H-inverse
$K_{10362} =$	0x41fda1a6e7439711	0x41fda1a6e7439711
$K_{10363} =$	0xeb8e4371692dd431	0xeb8e4371692dd431
$K_{10364} =$	0x514ad0208e26f978	0x514ad0208e26f978
$K_{10365} =$	0x942f489003b19b0d	0x942f489003b19b0d
$K_{10366} =$	0xefc2048fa4fa3896	0xefc2048fa4fa3896
$K_{10367} =$	0xda18dbf105d2b23b	0x22598a0bc31b468a
$K_{10368} =$	0x8ccfab7ff0afca67	0xa835104b88385f2a
$K_{10369} =$	0x55172839a159896d	0xe483c319004b291b
$K_{10370} =$	0x398faf4089a9db02	0xcc0f0138b485a586
$K_{10371} =$	0x577348385c0c1d21	0x30b7e35d353bf1a0
$K_{10372} =$	0x95b0a32a8083a620	0xe80bae538455dbdf
$K_{10373} =$	0x143c4ca1ba30c7d3	0x36398db6f148e058
$K_{10374} =$	0xaa7a5b344b4a096e	0xfa55bd23d0d64e47
$K_{10375} =$	0xdff2dc3cd876f034	0x4a931835045fcc02
$K_{10376} =$	0x25c66c83b3453a35	0xf727773d4acdff58
$K_{10377} =$	0x07aee857d191dc32	0xd095083ec0504952
$K_{10378} =$	0x381696484bbfc012	0xdb52a3dfd4257c4f

To the right of these keys that were to be used, we have written down the keys obtained by iteratively applying one-way function F, starting from the H-inverse value `0xdb5`$\cdots$`4f`. We can see that the left and right of the first 5 lines are equal, but that the two differ from the 6-th line onward. So the *true* but *useless* H-inverse point `0xdb5`$\cdots$`4f` allows the attacker to obtain correct keys up to the 4-th future time interval, but no further. An H-inverse is a *correct* inverse only if it follows true through the whole 16-steps. Occurrence of these unhelpful but true H-inverses should not be confused with *false alarms*, to be explained below. It is logically impossible to distinguish these unhelpful inverses from knowledge of the H-image point $y \in \hat{\mathcal{D}}$ alone.

C.2 Inverse Counts

For a random function acting on a space of size N, it is well-known [8], [23] that the number of k-th iterated image points is expected to be $(1-\tau_k)N$, where τ_k is recursively given by

$$\tau_0 = 0, \quad \tau_{k+1} = \exp(\tau_k - 1), \tag{4}$$

as long as $k \lll N$. Hence, assuming F to be a random function, given an output from Algorithm 2, i.e., an x satisfying $H(x) = y$ for some disclosed key y, we can expect it to be a *correct* answer with probability

$$(1-\tau_{16}) \sim 0.10654. \tag{5}$$

Hence, were Algorithm 2 to return η pre-images under H, the success rate would be

$$p = 1-(1-(1-\tau_{16}))^{\eta} = 1-(\tau_{16})^{\eta}.$$

Equating this with (3), we get

$$\eta \sim 9.14301 \tag{6}$$

as the expected number of "x maps to y" printouts from Algorithm 2. A more accurate lengthy in-depth estimation of this value shows it to be closer to 17.5, and is explained in Appendix D. Note that this is not related to the success rate of our attack, but concerns how many times the attacker needs to try out forged messages.

C.3 Relative Randomness of Sets

There is one tricky point that we should be careful about. For our success probability discussion of Section 3.4.1 to be valid, the set $\check{\mathcal{D}}$ and family $\check{\mathcal{H}}$ have to be taken randomly, or at least independently of each other.

While, as long as F is random, the chains in $\check{\mathcal{D}}$ and $\check{\mathcal{H}}$ may both be thought of as providing random choice of values in themselves, if the chains constituting the two sets were created in the same manner, $\check{\mathcal{D}}$ and $\check{\mathcal{H}}$ would not be random with respect to each other. Overlap of one element would automatically bring about overlap of multiple elements. On the other hand, the elements may be seen as having been *grouped together* and overlap is less likely to happen than is with two sets chosen truly independently at random. This problem was taken into account in our algorithms by adding the permutation P to the Hellman table chain, so that the two types of chains are created independently of each other.

C.4 False Alarms

False alarms occur because $\tilde{H}$ is not injective. These happen when Algorithm 2 encounters a merge between a Hellman chain and another $\tilde{H}$-chain that started from some point other than the starting point of the Hellman chain. As these require one to recreate the corresponding chain from start before it can be dismissed, it brings about extra work not accounted for during consideration of the online complexity. It is known that in the rainbow table method [24], false alarms take a considerable portion of the online time. In fact, the recent paper [2] is an effort at reducing this wasted time.

These false alarms are more rare in our case, because we are not working with a *complete* table. It is reflected in our experiment result, where only about 0.4% of the target data we tried brought about false alarms, as in Appendix B. If under some other setup of μTESLA, should these false alarms cause any problem with our online phase operating in realtime, then the *check point* idea from [2] can be applied to our Hellman table.

APPENDIX D INVERSE COUNTS

This section discusses the issue of how often *useless* inverses, explained in Section C, are encountered. Our first approximation, as given by (6), showed that, on average, 9.14 inverses are expected from Algorithm 2. But this approximation disregards the fact that probability distribution of image points is not uniform.

Consider the directed graph associated with a random mapping F acting on a finite set $\mathcal{N}$ of size N. It is known [8] that the ratio of i-node points, that is, the number of points with i inverses, is expected to be

$$p_i = \frac{1}{e} \cdot \frac{1}{i!}, \tag{7}$$

as long as the set size N is large compared to i. It is instructive to note that the ratio of points with no inverses, i.e., $p_0 = \frac{1}{e}$, agrees with τ_1, as given by (4).

Denote by $F_n = F \circ \cdots \circ F$, the mapping obtained through doing n iterations of F and define

$$\mathcal{N}_{i,n} = \{x \in \mathcal{N} \mid \#\big(F_n^{-1}(x)\big) = i\}$$

to be the set of i-nodes for n iterations of F. We shall denote the expectation for F_n i-node ratio by

$$p_{i,n} = \#(\mathcal{N}_{i,n})/N.$$

We have already stated the values $p_{i,1}$ as p_i above. Given any n, since the ratios should add up to one, it is clear that $\sum_{i=0}^{\infty} p_{i,n} = 1$, and since the sets $F_n^{-1}(\mathcal{N}_{i,n})$ $(i = 0, \ldots, \infty)$ are disjoint and cover the whole space $\mathcal{N}$, it is also clear that $\sum_{i=0}^{\infty} i \cdot p_{i,n} = 1$.

To continue our discussion, we set

$$f_n(x) = \sum_{k=0}^{\infty} p_{k,n} x^k$$

and employ the generating function technique. In this language, the two relations concerning $p_{i,n}$ we have just discussed can be written as $f_n(1) = 1$ and $f_n'(1) = 1$, respectively.

We are more interested in the following value, whose calculation is somewhat tedious, but not difficult.

Lemma 2: The F_n i-node ratio values $p_{i,n}$ satisfy the equation

$$\sum_{i=0}^{\infty} i^2 \cdot p_{i,n} = n + 1. \tag{8}$$

Proof: We already know $f_n'(1) = 1$ for all n, and since

$$\sum_{i=0}^{\infty} i^2 \cdot p_{i,n} = \sum_{i=0}^{\infty} i \cdot p_{i,n} + \sum_{i=0}^{\infty} i(i-1) \cdot p_{i,n} = f_n'(1) + f_n''(1),$$

proving the above statement is equivalent to showing $f_n''(1) = n$, for all n.

The $n = 1$ case can be explicitly calculated after substitution of (7), providing for the starting point of our induction on n.

Now, it should not be too hard to convince oneself that

$$p_{i,n+1} = \sum_{j=0}^{\infty} p_j \Big(\sum_{k_1+\cdots+k_j=i} p_{k_1,n} p_{k_2,n} \cdots p_{k_j,n} \Big),$$

with appropriate interpretation taken for empty products and the $i = 0$ case. Next, notice that coefficient of x^i in $f_n(x)^j$ may be expressed as

$\sum_{k_1+\cdots+k_j=i} p_{k_1,n}p_{k_2,n}\cdots p_{k_j,n}$, so that

$$\begin{aligned} f_{n+1}(x) &= \sum_i p_{i,n+1}x^i \\ &= \sum_i\sum_j p_j\Big(\sum_{k_1+\cdots+k_j=i} p_{k_1,n}p_{k_2,n}\cdots p_{k_j,n}\Big)x^i \\ &= \sum_j p_j \sum_i\Big(\sum_{k_1+\cdots+k_j=i} p_{k_1,n}p_{k_2,n}\cdots p_{k_j,n}\Big)x^i \\ &= \sum_j p_j\cdot f_n(x)^j. \end{aligned}$$

Using this, we can follow through the next set of equalities to complete the induction step.

$$\begin{aligned} f''_{n+1}(1) &= \Big[\frac{d}{dx}\frac{d}{dx}\Big(\sum_j p_j\cdot f_n(x)^j\Big)\Big]_{x=1} \\ &= \Big[\frac{d}{dx}\frac{d}{dx}\Big(\sum_{j=0}^{\infty}\frac{1}{e}\cdot\frac{1}{j!}\cdot f_n(x)^j\Big)\Big]_{x=1} \\ &= \frac{1}{e}\Big[\frac{d}{dx}\frac{d}{dx}\exp\big(f_n(x)\big)\Big]_{x=1} \\ &= \frac{1}{e}\Big[\frac{d}{dx}\Big(f'_n(x)\cdot\exp\big(f_n(x)\big)\Big)\Big]_{x=1} \\ &= \frac{1}{e}\Big(f''_n(1)\cdot\exp(f_n(1)) \;+\; (f'_n(1))^2\cdot\exp(f_n(1))\Big) \\ &= \frac{1}{e}\Big(n\cdot e+1\cdot e\Big) \;=\; n+1. \end{aligned}$$

The induction hypothesis was used in the last equality, and we have shown $f''_n(1) = n$ for all n, proving our original statement. □

We know from Lemma 1, the probability for two sets to have at least one common element. Let us see how many of these common points we would find on average.

Lemma 3: Suppose a set $\mathcal{D}$ of D distinct points and a family $\mathcal{H}$ of H (not necessarily distinct) points, both chosen uniformly at random from $\mathcal{N}$, a set of size N, are given. If $D \lll N$, we can expect $\mathcal{D}$ and $\mathcal{H}$ to contain $\frac{DH}{N}$ elements in common.

Proof: The probability $p(k)$ that $\mathcal{D}$ and $\mathcal{H}$ contain k elements in common can be written as

$$p(k) = \binom{H}{k}\cdot\Big(1-\frac{D}{N}\Big)^{H-k}\cdot\Big(\frac{D}{N}\Big)^k$$

for $k \le H$. For $k > H$, we have $p(k) = 0$.

We can approximate this as

$$p(k) \;\sim\; \binom{H}{k}\cdot\exp\Big(-\frac{D(H-k)}{N}\Big)\cdot\Big(\frac{D}{N}\Big)^k,$$

under the assumption that $D \lll N$. Let us approximate this once more as

$$\begin{aligned} p(k) &\sim \frac{H^k}{k!}\cdot\exp\Big(-\frac{DH}{N}\Big)\cdot\exp\Big(\frac{D}{N}\Big)^k\cdot\Big(\frac{D}{N}\Big)^k \\ &\sim \exp\Big(-\frac{DH}{N}\Big)\cdot\frac{1}{k!}\cdot\Big(\frac{DH}{N}\cdot\exp(\frac{D}{N})\Big)^k, \end{aligned}$$

which is valid in the $k \lll H$ range. Finally, using this, the expectation for the number of common elements can be calculated as

$$\begin{aligned} &\sum_k k\cdot p(k) \\ &\qquad\sim\; \frac{DH}{N}\cdot\exp\big(\frac{D}{N}\big)\cdot\exp\Big(\big(\frac{DH}{N}\big)\cdot\big(\exp(\frac{D}{N})-1\big)\Big) \\ &\qquad\sim\; \frac{DH}{N}, \end{aligned} \tag{9}$$

where the second approximation follows from $D \lll N$.

Notice that even though our approximation is valid only for $k \lll H$, the $\frac{1}{k!}$ factor of $p(k)$, which approaches zero quite quickly, covers up this discrepancy, so the validity of the final outcome (9) should not be of question. □

We next combine the above two lemmas.

Lemma 4: Suppose a set $\check{\mathcal{D}}$ of D points and a family $\check{\mathcal{H}}$ of H points, both chosen uniformly at random from $\mathcal{N}$, a set of size N, is given. Assuming elements of $\hat{\mathcal{D}} := F_n(\check{\mathcal{D}})$ to be distinct and $D \lll N$, we can expect $\hat{\mathcal{D}}$ and $\hat{\mathcal{H}} := F_n(\check{\mathcal{H}})$ to contain $\frac{DH}{N}\cdot(n+1)$ elements in common.

Proof: By choosing D and H elements $\check{\mathcal{D}}$ and $\check{\mathcal{H}}$ uniformly at random from the domain space, for each $i \ge 0$, we have effectively chosen $i\cdot p_{i,n}\cdot D$ and $i\cdot p_{i,n}\cdot H$ elements of $\hat{\mathcal{D}}\cap\mathcal{N}_{i,n}$ and $\hat{\mathcal{H}}\cap\mathcal{N}_{i,n}$ at random from the set $\mathcal{N}_{i,n}$, a subset of the range space, of size $p_{i,n}\cdot N$.

Through Lemma 3, the number of elements common to $\hat{\mathcal{D}}\cap\mathcal{N}_{i,n}$ and $\hat{\mathcal{H}}\cap\mathcal{N}_{i,n}$ can be approximated by

$$\frac{(i\cdot p_{i,n}\cdot D)(i\cdot p_{i,n}\cdot H)}{p_{i,n}\cdot N} = i^2\cdot p_{i,n}\cdot\frac{DH}{N}.$$

The total number of elements common to $\hat{\mathcal{D}}$ and $\hat{\mathcal{H}}$ can now be calculated as

$$\sum_i i^2\cdot p_{i,n}\cdot\frac{DH}{N} = (n+1)\cdot\frac{DH}{N},$$

where we have applied Lemma 2 to bring about the equality. We have shown $\frac{DH}{N}\cdot(n+1)$ to be the number of intersection expectancy in the image space. □

Finally, we can apply this lemma to the situation considered in the main body of this paper. Substituting $D \sim 2^{24.04}$ from (1), $H = t\cdot m = 2^{40}$, $N = 2^{64}$, and $n = 16$, we conclude that 17.495 *true alarms* can be expected from Algorithm 2.

This completes the purpose of this appendix section, but since we have the tools ready, we provide the following proposition as an interesting result.

Proposition 1: Suppose a set $\check{\mathcal{D}}$ of D points and a family $\check{\mathcal{H}}$ of H points, both chosen uniformly at random from $\mathcal{N}$, a set of size N, is given. Assuming elements of $\hat{\mathcal{D}} := F_n(\check{\mathcal{D}})$ to be distinct and $D \lll N$, the probability of $\hat{\mathcal{D}}$ intersecting with $\hat{\mathcal{H}} := F_n(\check{\mathcal{H}})$ nontrivially is approximately

$$1-\exp\Big(-\frac{DH}{N}\Big)^{n+1}.$$

Proof: By choosing D and H elements of $\check{\mathcal{D}}$ and $\check{\mathcal{H}}$ uniformly at random, for each $i \geq 0$, we have effectively chosen $i \cdot p_{i,n} \cdot D$ and $i \cdot p_{i,n} \cdot H$ elements of $\hat{\mathcal{D}} \cap \mathcal{N}_{i,n}$ and $\hat{\mathcal{H}} \cap \mathcal{N}_{i,n}$ at random from the set $\mathcal{N}_{i,n}$ of size $p_{i,n} \cdot N$.

Through Lemma 1, the probability for $\hat{\mathcal{D}} \cap \mathcal{N}_{i,n}$ and $\hat{\mathcal{H}} \cap \mathcal{N}_{i,n}$ to contain no element in common can be approximated by

$$\begin{aligned}\bar{p}_{i,n} &= \exp\Big(- \frac{(i \cdot p_{i,n} \cdot D)(i \cdot p_{i,n} \cdot H)}{p_{i,n} \cdot N} \Big) \\ &= \exp\Big(- i^2 \cdot p_{i,n} \cdot \frac{DH}{N} \Big).\end{aligned}$$

The probability of $\hat{\mathcal{D}}$ and $\hat{\mathcal{H}}$ not intersecting can now be calculated as

$$\bar{p}_n = \prod_{i=0}^{\infty} \bar{p}_{i,n} = \exp\Big(- (\sum_{i=0}^{\infty} i^2 \cdot p_{i,n}) \cdot \frac{DH}{N} \Big).$$

Lemma 2 can now be applied to bring about our conclusion. □

This proposition may not be interesting for $n > 1$, due to the lack of application, but the $n = 1$ case shows that the success probability of the pre-computation techniques have so far been underestimated. The usual approximation that does not take into account the non-uniform distribution of image points would have stated this value to be $1 - \exp(-\frac{DH}{N})$ rather than $1 - \exp(-\frac{DH}{N})^2$, which is slightly larger.

Appendix E
Computation and Storage of Base Stations: Comparisons

The efficiency advantages of X-TESLA for base stations may come from the extendable management of short chains in two levels. Fig. 11 compares the number of keys required for various μTESLAs, and shows that X-TESLA, which is DoS resistant as well as extendable, is the most flexible method for base station deployment when efficiency is considered. For multi-level μTESLA, the DoS resistant versions are compared since X-TESLA is also DoS resistant in the same context. For X-TESLA, the pre-computation of future chains required for commitment distribution has been counted additionally. The total computation of X-TESLA is 0.39% larger than that of μTESLA, but the pre-computation and storage requirements are 99.95% smaller than μTESLA. Note that the frequent regeneration of chains can also add to security.

To cover 1.7 years with 200ms intervals and 64-bit keys, the original μTESLA requires pre-computation of a single 2^{28}-key chain and 2.1GB of storage. Its recomputation version may pre-compute the whole chain but store only a decimated fraction of the keys $(2 \cdot 2^8 + 2^{20})$. This allows a tradeoff between storage and online recomputation load. The amount of pre-computation for DoS resistant multi-level μTESLA, is slightly more than that of μTESLA, with pre-computed CDMs demanding additional storage[17]. We consider 2-level μTESLA with 2^{14}-key chains, 3-level μTESLA with a 2^8-key chain for the highest level and 2^{10}-key chains for the rest, and 4-level μTESLA with 2^4-key chains for the highest and 2^8-key chains for the rest. With X-TESLA using 2^8-key chains and 2^{16}-key chains, three upper level chains and two lower level chains are necessary at runtime. So the required amount of pre-computation (and storage) with X-TESLA is only about 0.05% of those of μTESLA and DoS-resistant multi-level μTESLA. The amount of runtime storage is about 12.57% of that of even the recomputation version of μTESLA. As for the amount of real-time computation done at a base station, X-TESLA needs to generate a single $(2^8 + 2^{16})$-key chain every 3.6 hours. Similarly, the recomputation version of μTESLA should recompute 2^{16} keys within the same 3.6 hours.

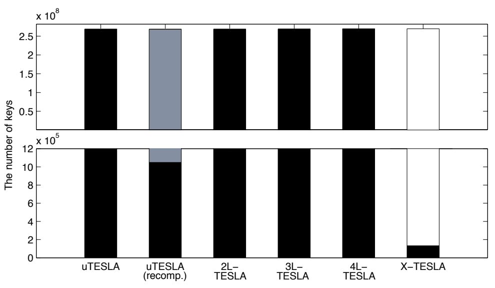


Fig. 11: Number of keys for covering 1.7 years with 200ms intervals. (The full bars, almost indistinguishable from each other in height, give the total number of keys. The number of pre-computed (black+gray) and stored (black) keys are also given. The recomputation version assumes keys recomputed (gray) in 2^8-key segments.)

Appendix F
Buffer Requirements under Harsh Conditions

Following the notation of Section 5.2.2, the various probability estimates given therein are valid only when r' is not too small, as they assume at least one proper Type 4 or CDM packet arriving at the sensor nodes. To treat the $r' \lll 1$ situation more properly, we may treat r' as the probability of this arrival[18], and a better estimate for X-TESLA would be

$$P_{\mathrm{X}} = 1 - \Big(1 - \min\Big\{\frac{B}{f'+1}, 1\Big\}\Big)^{2^8 r'}.$$

Corresponding probability $P_{2\mathrm{T}} = 1 - \binom{2^8 f'}{B} / \binom{2^8 (f'+r')}{B}$ for 2-level DoS tolerant μTESLA remains unchanged, where the numerator is taken to be zero when $2^8 f' + 1 \leq B$. For the 4-level version, such an estimation does not seem to be easy, but, in fact, because no CDM would be arriving

17. With DoS resistant M-level μTESLA, $(N^{M+1} - N)/(N-1)$ keys and $(N^M - N)/(N-1)$ CDMs must be pre-computed when each chain has N keys [19]. The extra storage required for pre-computed hash of CDMs is already larger than the whole runtime storage of X-TESLA.

18. This is related to the classical occupancy problem, and is approximately correct under the given assumption.

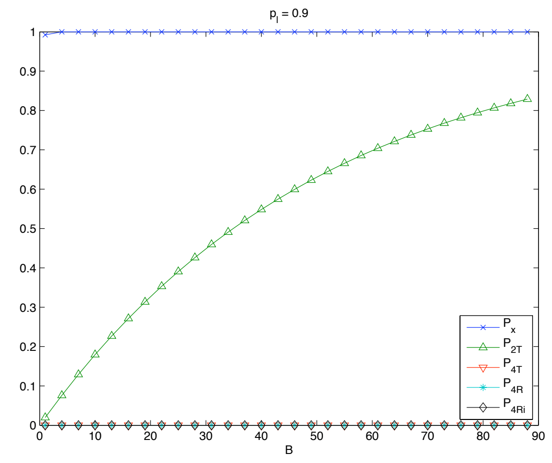


Fig. 12: Probability of proper Type 4/CDM handling with buffer size B, under harsh conditions. $f = 100$, $r = 2$, and $p_l = 0.9$.

in most of the 2^8-key intervals, we can expect $P_{4\mathrm{T}}$ to be practically zero. The same argument holds true for the DoS resistant version and we would have $P_{4\mathrm{Ri}} \sim 0$ and $P_{4\mathrm{R}} \sim 0$. These are all depicted in Fig 12, from which it is evident that X-TESLA excels other μTESLA variants in this harsh environment also.

Appendix G Energy Consumption for Sensor Nodes

Let a random process $X(t)$ have an exponential distribution such that $X(t) = \lambda e^{-\lambda t}$, and let $E[X]$ be the expected value of $X(t)$, that is the average distance between two packet arrivals, with regard to a message rate λ. We take a *slot* to be the time interval for which a single lowest level key is valid. For example, with X-TESLA above, one upper level interval covering 2^8-key lower level chain is said to have 2^8 slots. Let α be the number of slots within an upper level interval, and let $\mathcal{T}$ be the total number of slots within the whole duration of the key chains, or usage lifetime. Then we define

$$C_{i,\alpha} = \begin{cases} E[X] & \text{when } \sigma_i \bmod \alpha \geq E[X], \\ \sigma_i \bmod \alpha & \text{otherwise,} \end{cases}$$

the number of slots requiring computation for verification of the i-th packet that has just arrived, where, $\sigma_i = i \cdot E[X]$. We assume that r legitimate CDMs and $\mathcal{F}$ forged CMDs are received in each interval for simplicity. Packet losses are also disregarded for the same reason. Using these notions, we can write the energy consumption $\mathcal{E}$ of each μTESLA variant as the sum of costs from the following operations: (1) key computation whose number is given in terms of $C_{i,\alpha}$, (2) MAC computation and message radio reception, both of which counts to $\left\lfloor \frac{\mathcal{T}}{E[X]} \right\rfloor$, and (3) buffering and verifying of both real and adversarial CDM or Type 3/Type 4 packets. We can then analyze the energy consumption $\mathcal{E}$ of each TESLA to be as follows, for $C_K = 0.477$, $C_M = 0.508$, $C_{M'} = 0.539$, $C_H = 0.569$, $C_R = 0.106$, and $C_{R'} = 0.114$ (mJ).

$$\mathcal{E}_{\mu} = \left\lfloor \frac{\mathcal{T}}{E[X]} \right\rfloor (C_K E[X] + C_M + C_R).$$

$$\begin{aligned}\mathcal{E}_{2\mu} = {} & C_K \sum_{i=1}^{\left\lfloor \frac{\mathcal{T}}{E[X]} \right\rfloor} C_{i,2^{14}} + \left\lfloor \frac{\mathcal{T}}{E[X]} \right\rfloor (C_M + C_R) \\ & + \left\lfloor \frac{\mathcal{T}}{2^{14}} \right\rfloor \{C_K + C_M \cdot \min(\frac{2^{10}\mathcal{F} + 2^{10}r}{2^{10}r + 1}, B) \\ & + C_R \cdot (2^{10}\mathcal{F} + 2^{10}r)\}.\end{aligned}$$

$$\begin{aligned}\mathcal{E}_{2\mu'} = {} & C_K \sum_{i=1}^{\left\lfloor \frac{\mathcal{T}}{E[X]} \right\rfloor} C_{i,2^{14}} + \left\lfloor \frac{\mathcal{T}}{E[X]} \right\rfloor (C_M + C_R) \\ & + \left\lfloor \frac{\mathcal{T}}{2^{14}} \right\rfloor \{C_K + (C_{M'} + C_H + C_{R'}) \frac{2^{10}\mathcal{F} + 2^{10}r}{2^{10}r + 1}\}.\end{aligned}$$

$$\begin{aligned}\mathcal{E}_{3\mu} = {} & C_K \sum_{i=1}^{\left\lfloor \frac{\mathcal{T}}{E[X]} \right\rfloor} C_{i,2^{10}} + \left\lfloor \frac{\mathcal{T}}{E[X]} \right\rfloor (C_M + C_R) \\ & + \left\lfloor \frac{\mathcal{T}}{2^{10}} \right\rfloor \{C_K + C_M \cdot \min(\frac{2^6\mathcal{F} + 2^6 r}{2^6 r/2 + 1}, B) \\ & + \left\lfloor \frac{\mathcal{T}}{2^{20}} \right\rfloor \{C_K + C_M \cdot \min(\frac{2^{16}\mathcal{F} + 2^{16} r}{2^{16} r/2 + 1}, B) \\ & + C_R \cdot (2^6 \frac{\mathcal{F}}{2} + 2^6 \frac{r}{2})\} + C_R \cdot (2^{16} \frac{\mathcal{F}}{2} + 2^{16} \frac{r}{2})\}.\end{aligned}$$

$$\begin{aligned}\mathcal{E}_{3\mu'} = {} & C_K \sum_{i=1}^{\left\lfloor \frac{\mathcal{T}}{E[X]} \right\rfloor} C_{i,2^{10}} + \left\lfloor \frac{\mathcal{T}}{E[X]} \right\rfloor (C_M + C_R) \\ & + \left\lfloor \frac{\mathcal{T}}{2^{10}} \right\rfloor \{C_K + (C_{M'} + C_H + C_{R'}) \\ & \qquad (2^6 \frac{\mathcal{F}}{2} + 2^6 \frac{r}{2})/(2^6 r/2 + 1)\} \\ & + \left\lfloor \frac{\mathcal{T}}{2^{20}} \right\rfloor \{C_K + (C_{M'} + C_H + C_{R'}) \\ & \qquad (2^{16} \frac{\mathcal{F}}{2} + 2^{16} \frac{r}{2})/(2^{16} r/2 + 1)\}.\end{aligned}$$

$$\begin{aligned}\mathcal{E}_{4\mu} = {} & C_K \sum_{i=1}^{\left\lfloor \frac{\mathcal{T}}{E[X]} \right\rfloor} C_{i,2^8} + \left\lfloor \frac{\mathcal{T}}{E[X]} \right\rfloor (C_M + C_R) \\ & + \left\lfloor \frac{\mathcal{T}}{2^8} \right\rfloor \{C_K + C_M \cdot \min(\frac{2^4\mathcal{F} + 2^4 r}{2^4 r/3 + 1}, B) \\ & + C_R \cdot (2^4 \frac{\mathcal{F}}{3} + 2^4 \frac{r}{3})\} \\ & + \left\lfloor \frac{\mathcal{T}}{2^{16}} \right\rfloor \{C_K + C_M \cdot \min(\frac{2^{12}\mathcal{F} + 2^{12} r}{2^{12} r/3 + 1}, B) \\ & + \left\lfloor \frac{\mathcal{T}}{2^{24}} \right\rfloor \{C_K + C_M \cdot \min(\frac{2^{20}\mathcal{F} + 2^{20} r}{2^{20} r/3 + 1}, B) \\ & + C_R \cdot (2^{12} \frac{\mathcal{F}}{3} + 2^{12} \frac{r}{3})\} + C_R \cdot (2^{20} \frac{\mathcal{F}}{3} + 2^{20} \frac{r}{3})\}.\end{aligned}$$

$$\mathcal{E}_{4\mu'} = C_K \sum_{i=1}^{\left\lfloor \frac{\mathcal{T}}{E[X]} \right\rfloor} C_{i,2^8} + \left\lfloor \frac{\mathcal{T}}{E[X]} \right\rfloor (C_M + C_R)$$

$$+ \left\lfloor \frac{\mathcal{T}}{2^8} \right\rfloor \{C_K + (C_{M'} + C_H + C_{R'}) (2^4 \frac{\mathcal{F}}{3} + 2^4 \frac{r}{3})/(2^4 r/3 + 1)\}$$
$$+ \left\lfloor \frac{\mathcal{T}}{2^{16}} \right\rfloor \{C_K + (C_{M'} + C_H + C_{R'}) (2^{12} \frac{\mathcal{F}}{3} + 2^{12} \frac{r}{3})/(2^{12} r/3 + 1)\}$$
$$+ \left\lfloor \frac{\mathcal{T}}{2^{24}} \right\rfloor \{C_K + (C_{M'} + C_H + C_{R'}) (2^{20} \frac{\mathcal{F}}{3} + 2^{20} \frac{r}{3})/(2^{20} r/3 + 1)\}.$$

$$\mathcal{E}_{\text{x}} = C_K \sum_{i=1}^{\left\lfloor \frac{\mathcal{T}}{E[X]} \right\rfloor} C_{i,2^8} + \left\lfloor \frac{\mathcal{T}}{E[X]} \right\rfloor (C_M + C_R) + \left\lfloor \frac{\mathcal{T}}{2^{16}} \right\rfloor \{(C_M + C_K + C_R) + \sum_{i=1}^{2^8-1} \left(\frac{\mathcal{F}}{\mathcal{F}+1}\right)^i \left(\frac{1}{\mathcal{F}+1}\right) \cdot i \cdot (C_M + C_K + C_R)\}.$$

Here, C_K, C_M, $C_{M'}$, C_H, C_R, and $C_{R'}$ are the costs for key computation, MAC computation for two blocks, MAC computation for three blocks, hash computation for five blocks, radio reception, and radio reception by the DoS resistant multi-level μTESLA[19], respectively. From the estimation of Mica-Z power consumption [30] and our experiments, these values were set as 0.477mJ, 0.508mJ, 0.539mJ, 0.569mJ[20], 0.106mJ, and 0.114mJ, respectively, and applied to our analysis. Fig. 8-(a) shows that both X-TESLA and 4-level μTESLA are superior to others unless there are forged packets (coming from DoS attacks). If there are forged packets, X-TESLA consumes less energy than 4-level μTESLA, as in Fig. 8-(b). With X-TESLA, sensor nodes can skip unnecessary key computation and commitment verification to save energy.

19. DoS resistant multi-level μTESLA requires 17.9% larger ZigBee packets (due to the 24.1% larger 36-byte data) than X-TESLA to support immediate authentication of each commitment distribution. In our experiment, the CDMs of DoS resistant multi-level μTESLA cost around 8% more transmission time (5.4ms) than Types 3 and 4 packets of X-TESLA (5ms) over ZigBee channels, due to the additional 8-byte hash value inserted into each CDM.

20. RC5 is used for obtaining 8-byte hash value from 5 input blocks.