PAST : Probabilistic Authentication of Sensor Timestamps

Ashish Gehani Surendar Chandra
University of Notre Dame
E-mail: {ashish.gehani,surendar}@nd.edu *

Abstract

Sensor networks are deployed to monitor the physical environment in public and vulnerable locations. It is not economically viable to house sensors in tamper-resilient enclosures as they are deployed in large numbers. As a result, an adversary can subvert the integrity of the data being produced by gaining physical access to a sensor and altering its code. If the sensor output is timestamped, then tainted data can be distinguished once the time of attack is determined. To prevent the adversary from generating fraudulent timestamps, the data must be authenticated using a forward-secure protocol. Previous work requires the computation of n hashes to verify the $(n+1)^{th}$ reading. This paper describes PAST, a protocol that allows timestamps to be authenticated with high probability using a small constant number of readings. In particular, PAST is parameterized so that the metadata overhead (and associated power consumption) can be reduced at the cost of lower confidence in the authentication guarantee. Our protocol allows arbitrary levels of assurance for the integrity of timestamps (with logarithmically increasing storage costs) while tolerating any predefined fraction of compromised base stations. Unlike prior schemes, PAST does not depend on synchronized clocks.

1. Introduction

Sensor networks are designed to record information about the physical environment in which they are deployed. Hence, sensors must be located in close proximity to the target of measurement. This often precludes relying on wires for communication or external power. Instead, each sensor exchanges data wirelessly and relies on a battery. This limits the strength of the signals that they can transmit. As a result, a base station which communicates with the sensors must be situated in the immediate neighborhood. An adversary is therefore likely to be able to gain physical access to the sensors and base stations once they are deployed. With access, an adversary can generate spurious readings. This can have significant consequences, such as when the data is used for financial applications or as evidence in judicial proceedings. The vulnerability significantly impacts the viability of deploying sensors in a hostile environment. Our protocol records the time at which each sensor reading was generated. Once the time of an attack is known, we can distinguish which readings were generated before the sensor's integrity was compromised. The key is to ensure the soundness of the timestamping mechanism itself. This paper describes a protocol for *probabilistic authentication of sensor timestamps* (PAST) to address the issue.

PAST consists of three stages. Light weight cryptographic witnesses are generated for blocks of readings on board a sensor in the first stage. These witnesses provide partially trusted testimony to base stations that act as notaries. In the second stage, a notary gains increasing confidence in the timestamp of a block as it gets more consistent testimony from multiple witnesses. Once this process completes, the notary uses a forward-secure identity-based signature to certify the data in the third stage.

PAST ensures that the data stream generated by a sensor is distributed through multiple nodes in the network. As a result, only the sensor sees its own data stream in its entirety. This property is exploited to generate witnesses that other nodes can not fabricate as long as the number that collude does not exceed a predefined threshold. The witnesses used by a sensor are evolved at each time step. The protocol prevents an adversary from gaining any information about earlier witnesses if a sensor is compromised.

Since the data is certified with an identity-based signature, it can be verified in the absence of network connectivity. Further, since the signature is forward-secure, if a base station is compromised, the adversary cannot forge earlier timestamps. Finally, we use a new signature scheme that we have introduced in this paper. It allows the time-dependent verification key to be computed offline, unlike previous forward-secure identity-based signatures. Its security is equivalent to the underlying scheme on which it is based.

We analyze the security of the timestamps and data when an adversary is able to interfere with network connectivity, compromise sensors or base stations and attempt Sybil attacks. We show that PAST provides a high level of certainty for the timestamps with low storage and power consumption overhead. For example, when 7 witnesses are included in each PAST block and at least 4 must be verified before the block's timestamp is accepted, the probability that the block will be validated is 96% when as much as 35% of the network's base stations have been compromised. Fur-

*Supported by NSF Awards IIS-0515674 and CNS-0447671.

ther, this assurance is achieved while imposing only 0.01% storage and power consumption overhead.

Section 2 frames PAST's context and the constraint under which the protocol must operate. Section 3.2 explains how sensors can generate timestamps which can be probabilistically authenticated. Section 3.3 describes how base stations can certify the timestamps. PAST's security is analyzed in Section 4. Related work is described in Section 5. We draw our conclusions in Section 6.

2. Design Context and Constraints

The constraints on designing an authenticated timestamping protocol for a sensor network environment are described in this section.

2.1. Temporal Exposure of Signing Keys

If the data gathered from a sensor network is not authenticated, an adversary will be able to forge it without being detected. To prevent this, the data should be certified when it is being generated. This requires that the sensors are provisioned with cryptographic keys. Since sensors are deployed in large quantities, the budget for constructing each sensor is limited. This precludes using tamper-resilient enclosures for the sensors. (Secure smart-cards with limited protection cost $15 [4] while IBM's 4758 secure coprocessor, which is a more resilient device, costs $2,000 [1]. For many common sensors, these technologies can add significantly to the cost of each device.)

Hence, at first glance, the cost constraints would appear to rule out any guarantees for the sensed data once an adversary gains access to the cryptographc keys stored in the sensor. However, two related classes of algorithms target this problem. The first is forward-secure signatures [2], where the signing key is evolved periodically with a one-way function. If an adversary gains access to a signing key, they can not forge signatures for previously generated data since it is computationally prohibitively expensive to invert the one-way function.

The second class is intrusion-resilient signatures [8]. These require a remote node's participation in the process of updating the key in each new time interval. When a signing key is compromised, the appropriate remote node is notified to stop participating in the key update protocol. This prevents an adversary from computing new signing keys for future time periods. Therefore, they can not generate fraudulent future timestamps. Unfortunately, the use of intrusion-resilient signatures introduces a significant vulnerability in a sensor network setting. If an adversary wishes to prevent the data from a particular time period from being certified, they can flood the network link between the sensor and the base station during the preceding time period. This will prevent the key update protocol from completing within the allotted time.

The robustness to network noise, flooding and partitioning that we can achieve with a forward-secure scheme must be weighed against the self-correction property of intrusion-resilient signatures. We opt for the forward-secure scheme since it suffices for distinguishing untainted readings that were produced before a sensor was compromised from those that were subsequently generated.

2.2. Network Dependence of Verification

Checking the authenticity of a forward-secure signature requires access to the appropriate verification key. A certified version of the key can be included with the signed data. However, this introduces significant storage overhead (on the order of kilobytes for a signature that is computationally difficult to forge). Alternatively, when a node joins the network, it can broadcast a certified version of its verification key. Since each node on the network must either cache all the verification keys or query its neighbors for them when needed, the storage and communication overhead of such a scheme does not scale as the network grows in size. Instead, the third option is to register the verification key with a public key server. When a remote node needs the key, it can be retrieved from the server. Thereafter, the cached version can be utilized. However, this is only practical if remote nodes have reliable network connectivity with the server. If the remote node attempts to buffer data until it can be verified, an adversary can exploit this by disrupting communications till the buffer overflows and legitimate sensor readings must be discarded.

Identity-based signatures [15] allow arbitrary strings to be used as verification keys. The signing key can be derived from the verification key using a global secret parameter, known only to the administrator. In particular, a node's network address can be utilized as its verification key, obviating the need for key distribution. An identity-based signature can be made forward-secure by concatenating the signing time with the network address when constructing the verification key. However, this requires a new signing key to be derived by the administrator for each unique signing time. This would reintroduce dependence on the network. In Section 3.3 we construct a new forward-secure identity-based signature for which the verifier can evolve the new verification key without the intervention of the administrator. This will allow timestamps to be verified in the face of transient network connectivity.

2.3. Resource Constraints

Base stations act as gateways to which sensors send their data. Since each base station serves a large number of sensors, it is provisioned with substantial computational power, memory and bandwidth. Although this requires significant power consumption, a base station can still be deployed in the field using a renewable energy source, such as solar power, in conjunction with a rechargeable battery. (For example, a CerfCube [3] node can be powered with the 60-120 Watts generated by solar panels [10].)

A typical sensor has limited processing power and memory. The MICA1 mote's processor is an 8-bit microcontroller that runs at 4 MHz with 4 KB RAM, 128 KB

flash memory to store code and 512 KB to store data [7]. In principle this suffices to perform the modular exponentiation with large primes needed for asymmetric algorithms like RSA and Diffie-Hellman. However, despite careful selection of exponents and primes, the computations take a considerable amount of time to complete. Performing a single RSA operation with a small exponent and a key of size 512, 768 or 1024 bits takes 3.8, 8.0 or 14.5 seconds respectively [16]. If each sensor reading was to be signed, this would significantly limit the rate at which data could be generated. Batching the readings before generating signatures could amortize the cost and increase the rate at which signed sensor data could be generated. If a sensor reading is 16 bits and its timestamp is 16 bits, then at most 32 readings could be batched together in a single 1024-bit signature without adding a hash function to the computation. Even with amortization, this would require the mote to spend 0.5 seconds to authenticate a reading. Adding a hash would not only need a new function to be computed over all the data, but it would also require the data to be sent independently of the signed hash. Since the MICA1 mote transmits data at 40 Kb/sec, another 25.6 seconds would be needed to transmit the signed hash. These added costs will only be warranted if a large number of readings are batched together. However, the size of the non-volatile memory limits the number of readings that can be buffered. In addition, this may introduce more latency than is acceptable for sensor applications.

The most significant implication of using an asymmetric primitive is its effect on how long the sensor can be deployed in the field. Sensors must be deployed in the immediate vicinity of the target of measurement. This may be a location without any renewable energy sources. Therefore, each sensor must rely on the power that can be stored in its battery at the time of deployment. A MICA1 mote runs on 2 AA batteries which provide 2.5 Ampere-hours at 3 Volts. If a mote is performing a typical sensing operation, performing a computation and transmitting data, it uses 0.1 Watts [7]. At this rate, the mote will exhaust its batteries in 75 hours. Since the expected field life of a sensor is several months or years, the mote powers components down when it is not performing operations. In sleep mode, the MICA1 draws only 30 micro-Watts. In order for a mote to survive for long periods, data processing must be minimized. Symmetric cryptography primitives require much less computation. Encrypting a 29 byte packet on a MICA2 mote with TinySec [9] takes 2 milliseconds and computing its MAC (message authentication code) using a hash takes 3 milliseconds [11]. This is several orders of magnitude faster than the state of the art implementation of asymmetric cryptography primitives for motes - Sizzle's [6] assembly language elliptic curve operations. As a result, PAST only uses symmetric cryptography on the sensors.

2.4. Trust Model

Sensor networks must tolerate compromised nodes, collusion between nodes and transient network partitions. They can ameliorate the problems using cryptography and redundancy.

Since sensors and base stations are deployed in the field, some fraction of them may be compromised by an adversary. As a result, at any given time only some predefined fraction of these nodes is operating correctly. We model this by assuming that a randomly chosen node's probability of subverting the protocol is proportional to the fraction of nodes in the network that are currently compromised.

If an adversary gains access to multiple nodes, it can leverage the knowledge obtained from one node to disrupt the correct operation of another node. Therefore, when a node relies on information received from or sent to multiple other nodes, the trustworthiness of that data must account for the possibility that the remote nodes are colluding.

Sensor networks rely on low power wireless transmission for data exchange. Therefore an adversary can suppress the flow of information by generating radio interference. Although the network stack may provide reliable transport by retransmitting lost data, it can not ensure that delivery will occur within a fixed timeframe. Unpredictable delays in the receipt of data must be supported. In particular, a protocol that relies on receiving cryptographic material within fixed temporal windows is untenable in this setting.

Trust can be built using cryptography and by distributing the trusted computing base. While sensors are limited to using symmetric cryptography, base stations have sufficient resources to utilize asymmetric cryptographic primitives. Further, a sensor network consists of a large number of nodes. Sensors can exploit this by using a distributed protocol to prevent any one node from becoming a central point of failure. PAST composes these to provide the requisite assurance.

3. Protocol Description

We first provide a high level description of PAST in Section 3.1. Section 3.2 details how a sensor generates timestamped data. Section 3.3 describes how a base station can probabilistically authenticate the timestamps. Finally, Section 3.4 outlines how the base station certifies the data's timestamps.

3.1. Overview

PAST has three stages. The first occurs at the sensor where the reading is generated as depicted in Step 1 of Figure 1. As explained in Section 2.3, using an asymmetric cryptographic primitive to sign the data would use too much battery power. In fact, authenticating each reading individually would also consume a significant amount of power. Instead, readings are batched together in blocks. When enough readings have been buffered to populate a block, the data is encrypted with a symmetric cipher. The key used is not known to the base station to which the block will be transmitted. Instead it is the key of another base station in the network. This node will act as a *notary* by verifying the block's timestamp and then certifying it. Every base station

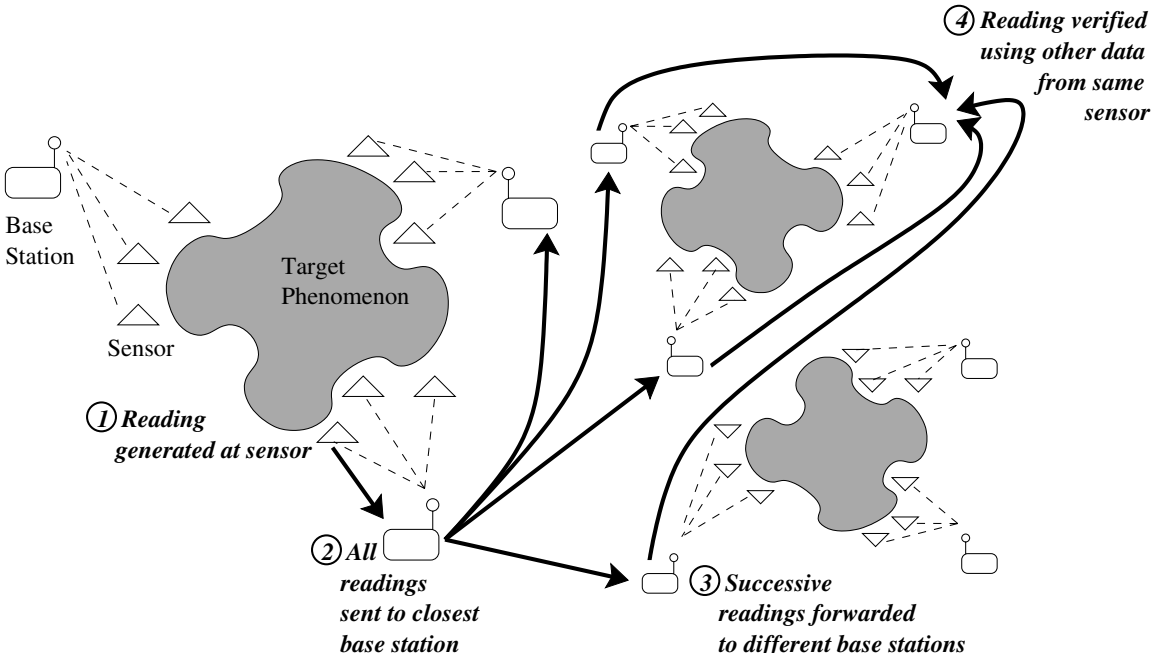

Figure 1. A set of witnesses are generated for each block of sensor readings in Step 1. The reading includes witnesses for other blocks. A block is sent to the base station closest to the sensor in Step 2. From there it is forwarded to another base station which acts as a notary in Step 3. As a notary, the base station uses witnesses from other blocks to certify a reading in Step 4.

in the network acts as a notary for some set of sensors distributed across the network. It will also act as a gateway for the set of sensors closest to it. The notaries act as impartial referees (unless they have been compromised by an adversary). To prevent a gateway from becoming a central point of failure, the set of sensors for which it is a notary is disjoint from the set of sensors for which it serves as a gateway. The notary stores a hash of the data keyed by the concatenation of the source sensor's address and the block's timestamp. It then encrypts the block with a storage server's public key and forwards it to long term storage. Successive blocks of sensor readings are forwarded to different notaries as illustrated in Step 3 of Figure 1. As a result, the stream of data generated by a sensor is not visible in its entirety to any other node in the network. PAST exploits this property by creating a set of *witnesses* using hashes of a sensor's most recently generated readings. No node other than the sensor can construct such a set since they do not have access to the entire data stream. (We assume that the private key of the storage server will not be compromised.) Each block of readings is prepended with such a set of witnesses and then transmitted to the closest base station in encrypted form. This completes Step 2 in Figure 1.

When a base station receives a data block from a sensor, it will be encrypted with a key unknown to it. However, the block's header will include the destination address of the base station that will serve as the notary for that block, as depicted in Figure 2. The gateway forwards the block to that base station, as shown in Step 3 of Figure 1, where the second stage of PAST can commence. When the notary receives the block, it extracts the source address of the sensor from the header. It uses this to lookup the decryption key needed to retrieve the sensor readings and the block's index, timestamp, hash, and set of witnesses. The hash of the reading is computed and compared to the one extracted from the encrypted block. If they do not match, the integrity of the block is compromised and it is discarded. Next, each witness must be checked. Since a witness is the hash of a previous block of data generated by the sensor, the base station where that block was sent will be able to validate the witness. The block's header, as shown in Figure 2, will include a notary's address prepended before each witness. This notary is the base station that can validate the witness. The index of the block whose hash it is can be calculated from the location of the witness in the header. The validating base station is contacted with the source sensor's address, the witness block's index and a nonce. It retrieves its record of the witness, combines it with the nonce and replies with the result. The notary combines its copy of the witness with the nonce and checks if this matches the result it received. As more witnesses are validated, the notary can be increasingly confident that the block's timestamp is authentic. The timestamp included in the block is that of the last reading. However, if each reading's timestamp is needed, it can be prepended before the reading and the timestamp in the header can be omitted. PAST's authentication guaran-

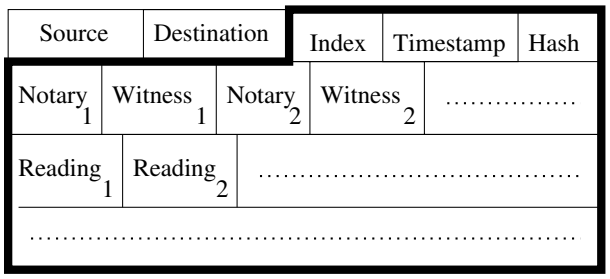

Figure 2. The format of a PAST block. The source sensor's address and destination base station's address are unencrypted. The rest of the data is encrypted. Successive blocks generated by the same sensor have consecutive index numbers. A set of witnesses and the notaries where they can be validated is included. The block includes as many readings as can be buffered in the sensor's memory.

tees will hold without any modification to the protocol.

In principle, the second stage of PAST suffices for authenticating the timestamps of the sensor readings. However, nodes running applications using the data may not be connected to the sensor network. This would prevent them from querying base stations to perform the second stage of PAST. The third stage of PAST accounts for this. In this final stage, after a notary has verified the testimony of multiple witnesses, it certifies the timestamp included with the block of readings using a digital signature. Recall that the base station does not have computational and power constraints. Therefore it can utilize asymmetric cryptographic primitives for this purpose. These signatures can be verified using the base station's public key, either at a node without connectivity to the sensor network or even offline. The notary encrypts the source sensor address, the timestamp and the block of readings in a single message using the public key of the storage server. This is done so that the data is not readable by an adversary once it leaves the notary. If it were, the adversary could use it to generate fraudulent witnesses for earlier sensor readings. The encrypted data is then signed with a forward-secure identity-based signature. If an adversary subsequently compromises the base station, access to the cryptographic keys will not enable it to generate a signature for an altered version of the encrypted data since the signature is forward-secure. Further, when an application uses the data, it can verify the signature without network access. This is because the signature is identity-based and the verification key evolution protocol introduced in Section 3.4 can operate offline.

3.2. Witness Generation

A sensor generates a stream of readings. PAST groups the data into timestamped blocks. The simplest authentication a sensor can provide is to include a token with each block that uses information known only to the sensor and the verifier. If the verifier is a base station, then when it is compromised it would be able to generate fraudulent readings. The verifier could claim that they came from the sensor and there would be no means of detecting the deception. The sensor could generate copies of each reading and send them to multiple destinations. However, this would consume proportionately more power and decrease the field life of the sensor. PAST addresses this issue by generating multiple short witnesses that can testify to the authenticity of the block.

A witness is a fixed size hash that is significantly smaller than the block it represents. Since it is small, transmitting copies of it to multiple base stations would only use a small amount of power. However, if the same witness was sent to each base station, any group of colluding nodes could all claim to have received an alternate fraudulent witness. Therefore, the witnesses must be distinct from each other. Simultaneously, they must attest to the integrity of the same block. As a result, they must be all have a verifiable relationship to the block. Composing the hash function multiple times would generate such witnesses. However, an adversary that has access to the first witness would be able to derive the subsequent ones by repeatedly hashing its witness. Thus the witnesses for a particular block can not use the same input value with the hash function composed a differing number of times.

In addition to the above constraints, the witnesses selected must support forward-secure authentication. Specifically, if a sensor is compromised, access to a current set of witnesses should not allow fraudulent earlier ones to be generated. This property is needed as the witnesses will be used to attest to the integrity of timestamps. Without this property earlier timestamps could be forged.

We now describe how PAST creates witnesses that satisfy the above constraints. The blocks of readings generated by a sensor are indexed, starting from 0 when the sensor is initially deployed. Subsequent blocks are consecutively numbered. If more witnesses are included in a block, it can be authenticated with greater certainty. However, as this number increases, so does the storage overhead and with it the power consumed to transmit the block. We parametrize the number of witnesses used for each block, terming it α. Each sensor maintains a FIFO of current witnesses. (The first-in first-out property is implemented as a circular array where the successor of the last element is the first. A pointer tracks the location of the current head of the FIFO.) The first entry is $h(r_i)$, the hash of r_i, the most recently generated block of sensor readings. (Throughout this paper $h()$ denotes a one-way hash function.) The second entry is $h^2(r_{i-1}) = h(h(r_{i-1}))$, the hash of the hash of r_{i-1}, the previous block of sensor readings. Successive entries of the FIFO are the result of composing the hash function an increasing number of times and using sequentially ear-

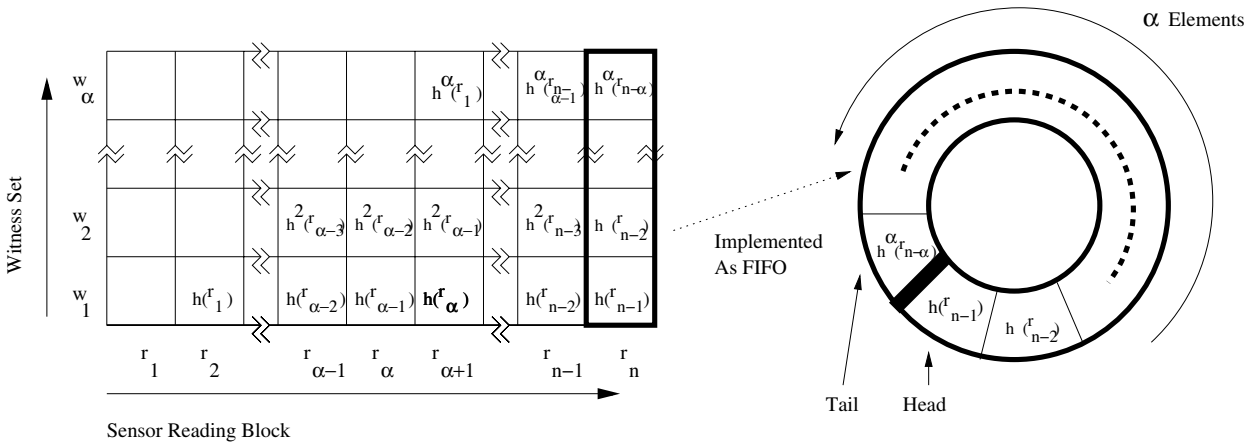

Figure 3. The current set of witnesses are stored in a FIFO implemented as a circular array. The column over r_i is the state of the witness FIFO when the i^{th} block of data is generated by the sensor. $h^j(r_i)$ represents the result of composing the hash function j times using r_i as the input.

lier blocks of readings as the input. Thus, the last entry is $h^{\alpha}(r_{i-(\alpha-1)})$. In practice, a FIFO entry has the form $\{b_i, h^j(r_i)\}$, where b_i is the base station to which the readings r_i were sent.

Each time the sensor readings buffer fills, it is transmitted to the base station. After this completes, the element stored at the location of the FIFO's head pointer is overwritten with the transmitted block's hash. The head pointer is then decremented by one (modulo its size, α), making the new hash the last in the circular array. Each of the other elements in the FIFO are replaced with the result of hashing themselves. Thus, the FIFO element $h^j(r_i)$ is replaced with $h^{j+1}(r_i) = h(h^j(r_i))$. b_i does not have to be updated. (r_i is the set of readings in block i and $h^{j+1}(r_i)$ is the $(j+1)^{th}$ witness. i and j are indexes modulo α.) Figure 3 illustrates the state of the FIFO over time as new blocks of readings are generated.

Since the hash function is assumed to be difficult to invert, if an adversary gains access to the FIFO, it can not use its contents to generate the previous values stored in the FIFO. The contents of the FIFO can not be generated without access to sensor data stream. Finally, every FIFO entry is related to the current block by the property that it is derived from one of α preceding blocks. Thus, the entries in the FIFO possess the properties needed to serve as witnesses.

3.3. Testimony Verification

A sensor shares a unique symmetric encryption key with each base station that acts as a notary on its behalf. At the time of deployment, the sensor is provisioned with keys to communicate with β base stations. A β-element array Ψ in the sensor stores entries of the form $\{b_v, k_v\}$ where b_v is a base station's address and k_v is its key. A pseudorandom generator is used to permute the array elements at initialization. Subsequently, after every β blocks have been transmitted, the array is randomly permuted again. This prevents an adversary from predicting which base stations will be designated as notaries for a given block of readings. As a result, the best an adversary can do is to compromise a random base station. Thus, when authenticating a block, the probability that a notary is compromised is uniformly μ_b.

When a set of readings are to be transferred to a base station, they are formatted into a block like the one depicted in Figure 2. First, the block's index is incremented by one. Then, the last reading's timestamp is stored. Next, the hash of the readings is computed. The contents of the witness FIFO are copied starting from the head. Finally, the next entry $\{b_v, k_v\}$ from the array Ψ is retrieved. k_v is used to symmetrically encrypt the aforementioned fields. Then the sensor's address, s, is entered in the source field and b_i in the destination field. Finally, the PAST block is transmitted to the closest base station which acts as a gateway and forwards the block to b_v. b_v decrypts it and commences verification.

b_v will attempt to verify each witness. This requires access to the block of readings used to generate each witness. However, these can not be used directly. If they were accessible, then an adversary could use the data to generate fraudulent witnesses. Instead PAST utilizes an interactive protocol between base stations. When b_v needs to validate the witness $\{b_i, h^j(r_i)\}$, it does the following. First, it sends a message to b_i with the source sensor's address, s, the index i of the block whose integrity is being validated, the position j of the witness in the header and a random nonce, x. Since b_i had previously received r_i from s, it can calculate $w = h^j(r_i)$. b_i then sends b_v the blinded witness, $h(w \oplus x)$. (The hash prevents an eavesdropper from gaining any information.) b_v computes $h(h^j(r_i) \oplus x)$ and checks if it matches w. If it does, the witness has validated the current block.

If b_v does not receive a response within a fixed amount of time, it aborts. This is necessary for three reasons. The first is that the network may be partitioned and the remote node may not be accessible. The second is that an adversary may control the remote node and be attempting to halt the validation protocol. The third reason is that the remote node may not yet have received r_i from s. The latter case can be addressed by b_v explicitly signaling to b_i how long it can wait for r_i to arrive.

As testimony from more witnesses is validated, the certainty of the block's integrity increases. However, it should be noted that even a single validated witness is a strong authenticator. The reason is that compromising a base station only allows an adversary to deny the validity of a legitimate witness. It can not use the control of the base station to falsely validate a witness since that would require computing a hash preimage, which is computationally prohibitively expensive.

3.4. Timestamp Notarization

Once the authentication process described in Section 3.3 completes, the timestamp t_i and block of sensor readings r_i are concatenated. This is then encrypted using a randomly generated encryption key k_c and symmetric cipher $S()$ to produce $e_i = S_{k_c}(t_i||r_i)$. k_c is encrypted with the storage server's public key k_s and an asymmetric cipher $A()$ to yield $k'_c = A(k_s, k_c)$. Such a scheme is necessary since asymmetric ciphers are significantly slower than symmetric ones. The hybrid protocol allows data to be encrypted at the rate supported by the symmetric cipher while being decryptable only with the storage server's private key. Since the key k_c is not reused, signing k'_c with a forward-secure signature suffices to distinguish data from different time periods. (If k_c were reused, then e_i would also need to be signed.) As described in Section 3.1, an identity-based digital signature scheme with offline verification key evolution is needed. This will allow timestamp and data integrity verification to occur in the face of transient connectivity or the presence of network partitions.

We first describe Shamir's original identity-based signature scheme [15]. After that we explain how we modify it to allow offline evolution of the verification key. First, an RSA modulus N (which is a product of two large primes) is chosen. A large prime p that is relatively prime to $\phi(N)$ is selected. ($\phi()$ is the Euler totient function.) h is a one-way hash function. $N, p, h()$ are global parameters and can be broadcast. Only the administrator knows the factorization of N. It can use this to derive a unique value g which satisfies $g^p = \eta \pmod N$, where η is the user's identity. g is only shared with the user η. Only knowledge of g allows a message to be signed such that η can be used as the key to verify the signature. η can sign the key k'_c by selecting a random number x, then computing $\gamma = x^p \pmod N$ and $\delta = g.x^{h(\gamma||k'_c)} \pmod N$. The signature $\{\delta, \gamma\}$ can be verified by checking whether $\delta^p \stackrel{?}{=} \eta.\gamma^{h(\gamma||k'_c)} \pmod N$.

To make the signature forward-secure, the identity can be replaced with the concatenation of the recipient η and the timestamp τ. Wherever η was used, $\eta||\tau$ is now used. However, this requires the signer to obtain a new signing key g_τ from the administrator for each distinct value of τ. One method to implement this is by contacting the administrator online each time a new signature is to be generated. The other possibility is that the user obtains signing keys for a large range of values of τ in advance, incurring significant storage overhead. The scheme we introduce below does not suffer from either of these drawbacks. (To see why the above scheme is forward-secure, note that once time τ passes, g_τ is discarded. When the node is compromised, signatures for earlier time periods can not be generated since the corresponding g_τ keys have already been deleted.)

Our scheme is initialized with the same global parameters $N, p, h()$ as the original scheme [15]. At the outset, a user with identity η is provided with the same key, denoted by g_0 instead of g. As before, $g_0^p = \eta \pmod N$. After time τ passes, the signer computes $g_{\tau+1} = g_\tau^2 \pmod N$ and then discards g_τ. Deriving g_τ from $g_{\tau+1}$ is as intractable as the factorization problem [14]. This ensures the forward-security of the signing key. Thus, if an adversary compromises a base station at time $\tau + 1$, they will not be able to obtain the signing keys for time τ or earlier.

The procedure for generating a signature must be modified. If γ is being created at time τ, it is constructed as $\gamma = x^{2^\tau p} \pmod N$. Similarly, when δ is created at time τ, it is defined to be $\delta = g_\tau.x^{2^\tau h(\gamma||k'_c)} \pmod N$. The signature for k'_c is now $\{\tau, \delta, \gamma\}$. Finally, the verification condition becomes: $\boxed{\delta^p \stackrel{?}{=} \eta^{2^\tau}\gamma^{h(\gamma||k'_c)} \pmod N}$

To see why this condition should hold, note that:

$$\begin{aligned}\delta^p &= (g_\tau.x^{2^\tau h(\gamma||k'_c)})^p \pmod N \\ &= (g_0^{2^\tau}.x^{2^\tau h(\gamma||k'_c)})^p \pmod N \\ &= (g_0.x^{h(\gamma||k'_c)})^{2^\tau p} \pmod N\end{aligned}$$

and:

$$\begin{aligned}\eta^{2^\tau}.\gamma^{h(\gamma||k'_c)} &= (g_0^p)^{2^\tau}.(x^{2^\tau p})^{h(\gamma||k'_c)} \pmod N \\ &= (g_0.x^{h(\gamma||k'_c)})^{2^\tau p} \pmod N\end{aligned}$$

Our scheme's security directly reduces to that of the original signature scheme. If an adversary can generate fraudulent signatures for our scheme, then they can do so for time $\tau = 0$. This would mean that they can generate fraudulent signatures for the original scheme.

4. Security Analysis

We analyze PAST's robustness in the face of attacks on network links, compromises of sensors and base stations and Sybil attacks.

4.1. Network Attack

Data generated by a sensor traverses three different types of network links. The first is between the sensor and the

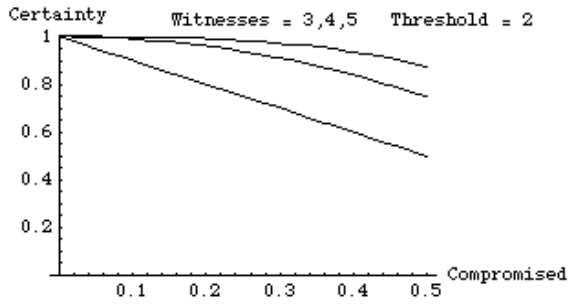

Figure 4. The x-axis represents the fraction of compromised base stations in the entire sensor network. The y-axis depicts the certainty with which a timestamp can be certified. The threshold for accepting the timestamp is 2, that is at least 2 witnesses must verified. The plots show the certainty of timestamp verification as 3, 4 or 5 witnesses are included in each PAST block.

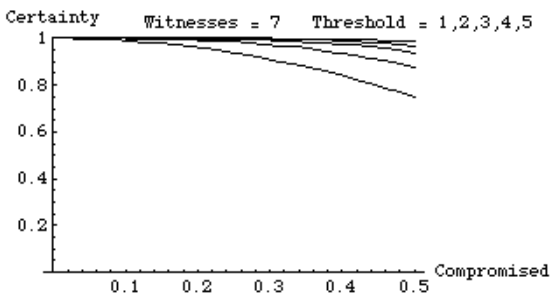

Figure 5. The x and y axes have the same semantics as in Figure 4. Each PAST block contains 7 witnesses. The threshold for validating a block's timestamp is varied from 1 to 5 witnesses.

closest base station that acts as a gateway. The second is the link between the gateway and the base station acting as a notary. The third is between the notary and the storage server. A PAST block includes a hash of the readings which serves as an integrity check. Except for the source and destination address, the rest of the block is encrypted. Thus, the integrity and confidentiality of the data is assured during the first two links, from the sensor through the gateway to the notary. Once a notary validates a PAST block, it encrypts the data with the storage server's public key and signs the result with its own key. Therefore the confidentiality, integrity and authenticity are ensured over the third link.

Base stations are assumed to have sufficient storage to buffer and retransmit data. Therefore an adversary that interferes with the second and third network link will only slow data transmission, not halt it. However, the first link is more vulnerable to attack. This is because the sensor must complete the transmission of a PAST block within a limited period of time. After that, it will either have to stop sampling the physical environment since it will not have sufficient memory to store new readings, or it will have to discard the PAST block. This problem can be addressed by splitting the memory between a current PAST block and one that is in the process of being transmitted. While this will double the metadata overhead, it will provide the sensor a large window of time within which it can retransmit the previous PAST block if an adversary is jamming the network link to the gateway.

4.2. Node Compromise

Since sensors and base stations are deployed in the field, an adversary may gain complete control over some of them. Once a sensor is compromised, it can be made to generate readings of an adversary's choosing. If the readings have fraudulent timestamps claiming they were generated in an earlier time period, PAST can detect this. This is ensured by evolving the authenticating witnesses with a one-way function each time a new PAST block is constructed. Similarly, when a base station is compromised, an adversary is limited because of the use of a forward-secure digital signature applied to all data sent to the storage server. If new data with old timestamps is sent, the inconsistency will be detected.

PAST distributes the attestation operations among multiple nodes. This allows it to tolerate the subversion of a fraction of the deployed base stations. As the fraction of compromised base stations in the network increases, the probability that a designated notary will choose not to validate a legitimate witness increases. This is depicted in Figure 4, where at least 2 witnesses must provide valid attestations for the timestamp to be accepted. The number of witnesses included in each PAST block is varied from 3 to 5. Each curve is associated with a different number of witnesses present in a PAST block. When the block contains more witnesses, there is a greater likelihood of finding 2 that are associated with uncompromised notaries that will validate them. This can be seen in Figure 4 since the plot for a witness count of 5 has uniformly higher certainty than the one where the witness count is 4 (and similarly the plot for 4 has higher certainties than the one for 3). Even with 35% of the network's base stations compromised, the timestamps can be validated with 96% probability using just 5 witnesses per PAST block, as can be seen from Figure 4.

The threshold for validating a block's timestamp is the number of witnesses that must be verified before it can be certified. In principle, a threshold of 1 should suffice since

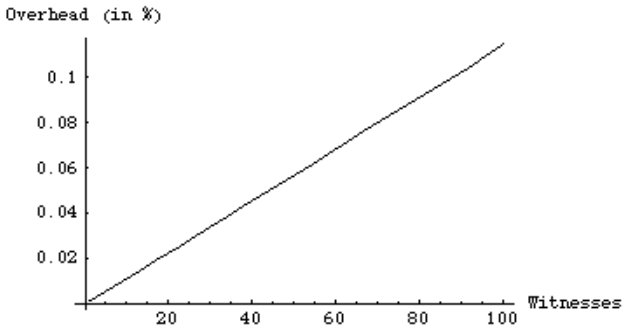

Figure 6. The storage overhead is plotted as a function of the number of witnesses included in a PAST block.

forging even a single witness would require computing the preimage of the hash function's output. However, the MAC used in TinySec is 32 bits [9] since it must be computed on a sensor. An adversary that compromises a sensor could send a stream of PAST blocks that are identical except for the use of differing values for the witnesses. In this manner, they could reduce the difficulty of computing a valid preimage. For example, if 256 variations of a PAST block are sent, then the complexity of forging a witness would drop to $2^{32-8} = 2^{24}$. If the threshold for accepting a timestamp is increased to 2 witnesses, then this attack's complexity increases quadratically. Thus, using a higher threshold is warranted when more assurance is required.

Figure 4 showed the benefit of increasing the number of available witnesses with a fixed threshold. In Figure 5, the PAST block is provisioned with 7 witnesses. The threshold is varied from 1 to 5. As the standard for validating a block is increased, there is a decrease in the certainty with which the timestamp can be certified. The curve for a threshold of 1 has the highest certainty, while the curve for a threshold of 5 has the lowest. Timestamps can be validated with high probability even if a stringent threshold is mandated. As seen in Figure 5, when 35% of the network's base stations are compromised and 4 witnesses must be verified out of 7 in a PAST block, the timestamp can still be validated with 96% probability.

The size of the memory in a sensor bounds the number of readings that can be buffered. As a result, there is a limit to the extent to which the storage overhead of the witnesses can be amortized by batching readings together. In the case of the MICA mote, the memory size is 512KB. A witness would be implemented using a TinySec MAC which is 4 bytes long [9] and the associated notary would be represented using a 2 byte TinySec address. As shown in Figure 6, PAST uses very little storage overhead for reasonable numbers of witnesses. If we used 7 witnesses to achieve the assurance of Figure 5, the storage overhead would be 0.01%. If a 100 witnesses were included in each PAST block, the storage overhead would be 0.11%. Storage overhead directly translates to power consumption since the metadata must be transmitted and communication requires a sensor to operate at its peak power level. PAST's utility derives from the fact that PAST imposes a small overhead while providing a high level of certainty in the sensor's timestamps.

4.3. Sybil Attack

To avoid a central point of failure, a protocol may require multiple nodes to participate. A Sybil attack [5] works around this by masquerading a single node as a set. The protocol can then be made to fail since the nodes it relies on are not independent and distinct. Such an attack is of particular significance in sensor network settings. The reason is that it is common for each sensor to be in communication range of a single base station. In such a case, when a sensor attempts to communicate with other nodes in the network, the gateway can effect a Sybil attack. It can masquerade as any remote node since all network traffic to and from the sensor passes through that gateway. PAST specifically guards against such attacks. All sensor output is encrypted using a key that is not known to the gateway. This prevents the gateway from masquerading as the originating sensor. A different key is used to encrypt a sensor's communication with distinct base stations. This prevents a notary from masquerading as a sensor.

5. Related Work

Bellare and Miner introduced the idea of forward-secure signatures in 1999 [2]. Przydatek, Song and Perrig subsequently framed the problem in the context of sensor networks [13]. As part of their work on secure information aggregation in 2004, they proposed the following scheme. Each sensor shares a key with a base station. The data generated by the sensor is hashed using this key. After a predefined period of time passes, a new shared key is computed by hashing the old one which subsequently discarded. The same operation is simultaneously performed on the sensor and the base station. When data from the n^{th} epoch needs to be authenticated, the n^{th} key is needed. Assuming the current epoch is later than the n^{th} epoch, there is no way to retrieve the earlier key. Therefore the verifier must be provisioned with the initial key and they must compose the hash function n times to derive the n^{th} key from the first. Since sensors are deployed in the field for long periods, n can grow large. Performing n hashes to verify each reading becomes expensive, making the scheme computationally unscalable. Alternatively, the verifier can pre-compute and store the composed hashes. However, this makes the scheme unscalable in terms of space usage. The paper acknowledges the scheme's deficiency and leaves its resolution as an open problem. PAST uses a small constant number of hashes to verify a single reading. This makes it scalable in terms of both time and space, at the cost of a small reduction in the probability of verification. Further, PAST does not rely on synchronized clocks.

Subsequent work on providing forward-secure authentication and secrecy focused on the problem of managing the keys [12]. It used the same hash chaining scheme as above. However, the first key is split using a polynomial-based secret sharing scheme. The pieces are then either distributed among the neighboring nodes of either a sensor or a data aggregation node. When a piece of sensor data is to be verified, a request goes to its source which then reconstitutes the first key. This is then used to compose the hash function n times if the data was the n^{th} reading and verify the value's integrity. The advantage of this scheme over previous work is that the verifier does not need to store and manage the keys used for checking sensors' output. Since nodes in the field have limited storage, composed values of the keys can not be stored. Therefore, this scheme must also rely on n hash compositions to verify the n^{th} reading. (PAST only needs to perform a small constant number of operations to verify a reading, rather than an unbounded sequence.) In addition, an adversary can use their verification protocol to force a node to reconstitute the first key at a time of the adversary's choosing. Since the nodes are deployed in the field, the adversary can subvert it at this point and get the first key. With this, they can forge past sensor data. PAST is not susceptible to such an attack since it does not store the original key indefinitely. Specifically, it is discarded after the first few sensor readings are generated.

6. Conclusion

We have described PAST, a probabilistic protocol for timestamping sensor data. It provides forward-secure timestamp authentication when sensor and base station cryptographic keys are compromised. It only requires a small constant number of operations to verify a reading. This improves over previous forward-secure sensor network protocols which required $O(n)$ hashes to verify the n^{th} reading in the data stream. Unlike previous schemes, PAST does not rely on synchronized clocks.

PAST ensures that the data stream generated by a sensor is distributed through multiple nodes in the network to avoid a central point of failure. It also prevents an adversary from gaining any information about earlier witnesses if a sensor is compromised. PAST provides a high level of confidence in the timestamps' validity while imposing a low storage and power consumption overhead.

References

[1] http://www.ibm.com/security/cryptocards/

[2] Mihir Bellare and Sarah Miner, A Forward-Secure Digital Signature Scheme, Advances in Cryptology, Lecture Notes in Computer Science 1666, 1999.

[3] http://www.intrinsyc.com/products/cerfcube/

[4] http://www.cyberflex.com

[5] John Douceur, The Sybil Attack, Proceedings of the 1st International Workshop Peer-to-Peer Systems, 2002.

[6] Vipul Gupta, Matthew Millard, Stephen Fung, Yu Zhu, Nils Gura, Hans Eberle and Sheueling Chang Shantz, Sizzle: A Standards-Based End-to-End Security Architecture for the Embedded Internet, Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications, 2005.

[7] Mike Horton, David Culler, Kris Pister, Jason Hill, Robert Szewczyk and Alec Woo, MICA - The Commercialization of Microsensor Motes, Sensors, 19(4), 2002.

[8] Gene Itkis and Leonid Reyzin, SiBIR: Signer-Base Intrusion-Resilient Signatures, Advances in Cryptology, Lecture Notes in Computer Science 2442, 2002.

[9] Chris Karlof, Naveen Sastry and David Wagner, TinySec: A Link Layer Security Architecture for Wireless Sensor Networks, Proceedings of the 2nd ACM Conference on Embedded Networked Sensor Systems, 2004.

[10] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler and John Anderson, Wireless Sensor Networks for Habitat Monitoring, ACM International Workshop on Wireless Sensor Networks and Applications, 2002.

[11] David Malan, Crypto for Tiny Objects, Harvard University Technical Report TR-04-04, 2004.

[12] Yi Ouyang, Zhengyi Le, James Ford and Fillia Makedon, Local Data Protection for In-Network Processing in Sensor Networks, IEEE International Conference on Pervasive Services, 2005.

[13] Bartosz Przydatek, Dawn Song and Adrian Perrig, SIA: Secure Information Aggregation in Sensor Networks, Proceedings of the ACM Conference on Embedded Networked Sensor Systems, 2003.

[14] Michael Rabin, Digital signatures and public-key functions as intractable as factorization, MIT Technical Report TR-212, 1979.

[15] A. Shamir, Identity-based cryptosystems and signature schemes, Advances in Cryptology, Lecture Notes in Computer Science 196, 1984.

[16] Ronald J. Watro, Derrick Kong, Sue-fen Cuti, Charles Gardiner, Charles Lynn and Peter Kruus, TinyPK: securing sensor networks with public key technology, Proceedings of the 2nd ACM Workshop on Security of Ad hoc and Sensor Networks, 2004.