Detecting Node Compromise in Hybrid Wireless Sensor Networks Using Attestation Techniques

Christoph Krauß, Frederic Stumpf[*], and Claudia Eckert

Department of Computer Science
Darmstadt University of Technology
Darmstadt, Germany
{krauss,stumpf,eckert}@sec.informatik.tu-darmstadt.de

Abstract. Node compromise is a serious threat in wireless sensor networks. Particular in networks which are organized in clusters, nodes acting as cluster heads for many cluster nodes are a valuable target for an adversary. We present two efficient hardware-based attestation protocols for detecting compromised cluster heads. Cluster heads are equipped with a Trusted Platform Module and possess much more resources than the majority of cluster nodes which are very constrained in their capabilities. A cluster node can verify the trustworthiness of a cluster head using the Trusted Platform Module as a trust anchor and therefore validate whether the system integrity of a cluster head has not been tampered with. The first protocol provides a broadcast attestation, i.e., allowing a cluster head to attest its system integrity to multiple cluster nodes simultaneously, while the second protocol is able to carry out a direct attestation between a single cluster node (or the sink) and one cluster head. In contrast to timing-based software approaches,the attestation can be performed even if nodes are multiple hops away from each other.

Keywords: Sensor Network, Security, Trusted Computing, Attestation.

1 Introduction

Wireless sensor networks (WSNs) [1] provide a technological basis for many different security-critical applications, such as military surveillance, critical infrastructure protection and surveillance. WSNs can be deployed in unattended and even hostile environments for monitoring the physical world. The monitored environment is covered by hundreds or even thousands of sensor nodes with embedded sensing, computation, and wireless communication capabilities. If sensor nodes are not specially protected, an adversary can easily compromise them, recover information (e.g. keying material) stored on the nodes, and subvert them to act as authorized nodes in the network to perform insider attacks.

One approach to detect compromised nodes is based on attestation techniques, where sensor nodes must prove that their system has not been modified

[*] The author is supported by the German Research Foundation (DFG) under grant EC 163/4-1, project *TrustCaps*.

by an adversary. Attestation techniques that have already been proposed for WSNs [2,3,4] are software-based and rely on relatively accurate time measurement. These techniques are unsuitable for attestation along multiple hops and when static interferences delay message transmissions, which prevents an exact time measurement. One promising approach for overcoming the shortcomings of software-based attestation is using the Trusted Platform Module (TPM) as specified by the Trusted Computing Group (TCG) [5] as the trust anchor for attestation protocols. The trust anchor is responsible for providing assurance of delivered attestation values. The TPM provides such a hardware-based trust anchor. It also offers certain cryptographic functions which provide the foundation for attesting the configuration of the local platform to a remote platform. Due to the large scale and desired low-cost of WSNs, it is not feasible to integrate a TPM into each individual node. Fortunately, many WSNs are organized in clusters where a minority of nodes perform some special functions. These nodes may act as cluster heads (CH), performing special duties, such as data aggregation or key management for a number of cluster nodes (CN).

Since CHs are a valuable target for an adversary, it might be reasonable to equip them with a TPM in scenarios where a high level of security is desired. CNs and the sink should be able to verify whether a CH is still trustworthy, even if it is multiple hops away. Since CNs are very limited in their resources, attestation protocols must be very lightweight, i.e., requiring only few, small messages and cheap operations (such as symmetric encryption).

In this paper, we propose two efficient TPM-based attestation protocols for hybrid WSNs organized in clusters. Networks consist of low-cost CNs and more expensive TPM-equipped CHs. The first protocol allows a number of CNs to simultaneously validate the trustworthiness of a CH in regular intervals, while the second protocol enables an individual CN (or the sink) to verify the trustworthiness of a CH at any time. Both protocols do not require expensive public key cryptography on the CNs and the exchanged messages are very short. Due to the unreliable, multihop communication, we can only prove the trustworthiness of the CHs, but not untrustworthiness. In addition, these protocols are not limited to cluster-based scenarios. For example the attestation protocols can be used in WSNs where many (mobile) TPM-equipped sinks exist, which are deployed in insecure locations. The network operator can verify if the data received from these sinks is still trustworthy.

2 Background on TCG-Mechanisms

The core of the TCG specifications [5] is the TPM, which is basically a smartcard, that serves as a trust anchor for trust establishment. The TPM offers protected storage for cryptographic keys and hardware enhanced calculation engines for random number generation, key-calculation and hash computation. Although the TPM chip was not specified as necessarily being tamper-resistant, many hardware vendors offer security mechanisms for preventing tampering and the unauthorized extraction of protected keys, such as active security sensors.

The TPM can generate and store cryptographic keys, both symmetric and asymmetric, and perform asymmetric cryptographic operations. The asymmetric keys can either be marked as migratable or non-migratable, which is specified when the key is generated. Non-migratable keys are always protected by the TPM and must not leave its protected storage.

The TPM also offers so-called Platform Configuration Registers (PCRs), which are used to store platform-dependant configuration values. These registers are initialized on power up and are used to store software integrity values. Software components (BIOS, bootloader, operating system, applications) are measured by the TPM before execution and the corresponding hash-value is then written to a specific PCR by extending the previous value:

$$Extend(PCR_N, value) = SHA1(PCR_N || value) \tag{1}$$

SHA1 refers to the cryptographic hash function used by the TPM and $||$ denotes a concatenation. The trust anchor for a so-called trust-chain is the *Core Root of Trust Measurement* (CRTM), which resides in the BIOS and is first executed when a platform is powered up. The CRTM then measures itself and the BIOS, and hands over control to the next software component in the trust-chain. For every measured component an event is created and stored in the Stored Measurement Log (SML). The PCR values can then be used together with the SML to attest the platform's state to a remote entity. To assure that these values are authentic, they are signed with a non-migratable key, the Attestation Identity Key (AIK). The remote platform can verify the signature and compare these values with reference values to see if the system integrity is trustworthy.

The TPM also offers a concept called *sealing*, which allows a data block to be bound to a specific platform configuration. A sealed message is created by selecting a range of platform configuration registers, a non-migratable key, and the data block which should be sealed. The TPM is then able to decrypt and transfer the sealed data block, only if its current platform configuration matches the platform configuration from when the sealing was executed. Sealing provides the assurance that protected messages are only recoverable when the platform is in a known system state.

3 Attestation Techniques

In this section we compare different attestation techniques and evaluate their applicability for WSNs.

3.1 TPM-Based Attestation

Existing attestation protocols [6,7] are based on the TPM's ability to report the system configuration to a remote party. These approaches are mainly developed for non-resource constrained computer systems and requires each communication partner to perform public key cryptography. The complete system configuration, as denoted in the PCRs of the attesting entity, must be transmitted to the verifying entity. The verifying entity evaluates the trustworthiness of the attested entity by comparing the received SML and PCR values with given reference values. Since the verifying entity receives the current platform configuration directly, we refer to this as *explicit attestation*. However, in hybrid WSNs most sensor nodes do not possess enough resources to perform public key cryptography and the transmission of large messages increases the energy consumption significantly. This causes explicit attestation to be inapplicable in WSNs.

To perform an attestation in WSNs, computation intensive operations must be transferred to nodes which posses sufficient computational power, and resource constrained sensor nodes need only to perform minimal verification computations. The sealing concept of the TPM enables an attestation without directly transferring the platform configuration (PCR values and SML). We refer to this as *implicit attestation*. This approach minimizes the amount of transmitted data and does not require public key cryptography on resource constrained nodes. Sealing provides the functionality to bind data to a certain platform configuration. The TPM releases, i.e., decrypts, this data only if the current platform configuration is valid. The disadvantage of this approach is that software updates change the values inside the PCRs. Since this results in inaccessible sealed data, this approach is not very applicable in non-resource constrained computer systems, where software configurations change very often through legitimate system updates. Fortunately, the software configuration of sensor nodes may not change during the whole lifetime of a WSN. Therefore, the attested entity is only able to decrypt a sealed data structure if the current platform configuration matches its initial platform configuration. Our protocols smartly exploit this property to enable a lightweight attestation of the trustworthiness of the attested entity.

3.2 Software-Based Attestation

The main disadvantage of TPM-based attestation is that the platform configuration only reflects the initial load-time configuration. Therefore, memory modifications during the runtime can not be detected, e.g., buffer-overflows. To overcome this shortcoming, attestation software may measure the memory and report the values to a remote party. In this case, the attestation software forms the trust anchor which must be protected against tampering. In [2,3,4], approaches based on measuring the execution time of an optimal attestation routine is introduced. The routine cannot be optimized further, i.e., the execution time cannot be made faster, which prevents an adversary from injecting malicious code without detection. However, the success of this approach relies critically on the optimality of the attestation routine and on minimal time fluctuations of the expected responses. Particularly in WSNs with multihop verification and external influences, time intervals for responses can vary. In these cases the attestation would fail, even though a sensor node is in a trustworthy system state.

In scenarios where attestation along multiple hops is required or external interferences prevent an exact time measurement, timing-based software attestation techniques are not applicable.

4 Setting and Notation

In this section we explain the setting and formulate the assumptions which are of concern for the protocols we propose.

4.1 Setting

We are considering a hybrid WSN, which is deployed in an unattended, hostile environment. The network consists of low-cost nodes and more expensive TPM-equipped nodes. TPM-equipped nodes act as cluster heads (CHs) for a number of low-cost cluster nodes (CNs), performing operations such as data aggregation, key management and so on.

We assume an adversary who tries to compromise a CH to access stored information, e.g., keying material, and misuse the node to perform insider attacks, e.g., injecting false reports to cause false alarms. Therefore, the adversary can try to read out data or re-program the node to behave according to the purposes of the adversary. Furthermore, due to wireless communication, the adversary can eavesdrop on all traffic, inject packets, or replay old packets.

CNs are very limited in their storage, computational, communication, and energy resources. However, they have enough space to store a few bytes of keying information and are able to perform some basic operations, such as computing hash functions, symmetric encryption, etc., but they are not able to perform public key cryptography. These nodes might be comparable to the Berkeley Mica Motes [8]. CHs are assumed to possess much more computing power, memory capacity, and energy resources, e.g., comparable to the resources of the Stargate platform [9]. The TPM, integrated in the CHs, is used to protect keys and other security related data. We do not require any modification of the TPM, such as adding support for symmetric encryption with external data. Since present TPMs only support internal symmetric encryption, some data must be stored temporarily in the Random Access Memory (RAM) of a CH for further processing. We assume that access to this temporarily stored data is not possible. As soon as future TPMs support symmetric encryption with external data this assumption can be revoked. To subvert a CH, an adversary must re-program and reboot the node to either modify the system so that access to the RAM is possible or to access the security related data directly. After a reboot with a modified system, the platform configuration is changed and the access to sealed data is no longer possible. Thus, this data is neither accessible directly to the adversary nor loaded into the RAM. To achieve the binding of cryptographic keys to a specific platform configuration, which subsequently prevents rebooting in a compromised system configuration, we assume that we have a reduced measurement architecture, such as IBM's IMA [10], that extends the trust chain specified by the TCG up to the firmware and therefore includes integrity measurement of the kernel and operating system of the CH.

Sensor nodes (CHs and CNs) can be deployed randomly, e.g., via aerial scattering. That means the immediate neighboring nodes of any sensor node are not known in advance. The sensed data is sent via multihop communication to the *sink*. The sink is assumed not to be constrained in its resources and cannot be compromised. It possesses all keying material shared with the sensor nodes.

4.2 Notation

CHs are denoted as $CH_i, i = 1, \ldots, a$ and the CNs are denoted as $CN_j, j = 1, \ldots, b$, where $b \gg a$.

$E(m, e)$ denotes the *encryption* of data m using an encryption function E and encryption key e. *Encrypted data* m using the key e is denoted with $\{m\}_e$. The *decryption* of $\{m\}_e$ using a decryption function D and the decryption key d is denoted with $D(\{m\}_e, d)$.

Applying a cryptographic *hash function* h on data m is denoted with $h(m)$. A one-way *hash chain* [11] stored on CH_i is denoted with $C^{CH_i} = c_0^{CH_i}, \ldots, c_n^{CH_i}$. The hash chain is a sequence of hash values of some fixed length l generated by a hash function $h : \{0,1\}^l \rightarrow \{0,1\}^l$ by applying the hash function h successively on a seed value $c_0^{CH_i}$ so that $c_{v+1}^{CH_i} = h(c_v^{CH_i})$, with $v = 0, 1, \ldots, n-1$.

A specific state of a CH_i is referred to as *platform configuration* $P_{CH_i} := (PCR_0, \ldots, PCR_p)$ and is stored in the appropriate PCRs of the TPM. Data m can be cryptographically bound to a certain platform configuration P_{CH_i} by using the `TPM_Seal` command. Using the `TPM_Unseal` command, the TPM releases, i.e., decrypts m only if the platform configuration has not been modified. This concept allows an implicit attestation to be performed without a direct validation of the PCRs by a CN. Since we are abstracting the `TPM_Seal` and `TPM_Unseal` commands, we denote our commands with `Seal` and `Unseal`. Given an non-migratable asymmetric key pair (e_{CH_i}, d_{CH_i}) we denote the *sealing* of data m for the platform configuration P_{CH_i} with $\{m\}_{P_{CH_i}}^{e_{CH_i}} = \mathtt{Seal}(P_{CH_i}, e_{CH_i}, m)$. To *unseal* data m it is necessary that the current platform configuration P'_{CH_i} is equal to P_{CH_i}: $m = \mathtt{Unseal}(P'_{CH_i} = P_{CH_i}, d_{CH_i}, \{m\}_{P_{CH_i}}^{e_{CH_i}})$.

5 Attestation Protocols

In this section we describe our two proposed protocols which enable a CN to verify the platform configuration of a CH. These protocols represent some basic primitives which can be used in conjunction or in more complex protocols. Our proposed protocols enable only CNs to verify the platform configuration of CHs. To verify the trustworthiness of received data from CNs, a CH has to perform additional mechanisms like redundancy checks or voting schemes.

We have adapted the sealing technique provided by the TPM to realize the implicit attestation (see Section 2). In the initialization phase the platform configuration of a CH is trustworthy. Data needed to perform a successful attestation is sealed in this phase to this platform configuration. Access to this sealed data is only possible if the CH is in the initial specified platform configuration. Compromising a CH results in a different platform configuration where access to this data is not possible. Thus, a successful attestation is no longer possible.

The first proposed protocol enables a broadcast attestation, where a CH broadcasts its platform configuration to its CNs in periodic intervals. This enables CNs to verify the platform configuration of the CH simultaneously. The second protocol enables a single CN (or the sink), to either individually verify the platform configuration of a CH using a challenge response protocol or to send data to a CH and receive a confirmation that the data has been received correctly and that the CH is trustworthy.

5.1 Periodic Broadcast Attestation Protocol (PBAP)

In some scenarios, many CNs perform measurements in parallel and in regular intervals. For example, a couple of CNs monitor the temperature in a specific region of the WSN. The measurement is performed every 10 minutes to see the change over time. Therefore, the CNs report their measurement nearly in parallel in specific time intervals to their CH. If each CN performs an individual attestation of the CH, this results in an avoidable overhead. It might be desirable that all CNs are able to nearly simultaneously verify if their CH is still trustworthy using an efficient mechanism.

The PBAP adapts the idea of μTESLA [12] to use one-way hash chains for authentication and extends it to enable attestation in hybrid WSNs. The sealing function of the TPM is used to bind a one-way hash chain to the platform configuration of a CH. A CH releases the values of the hash chain in periodic intervals, which can be verified by its CNs. The proof of trustworthiness of a CH is only possible while its platform configuration has not been modified.

The protocol is divided into two phases. In the *initialization* phase the CHs and the CNs are preconfigured before deployment. In the *attestation* phase, CHs periodically broadcasts an attestation message. This phase normally lasts for the whole lifetime of the CHs.

Initialization. Before CH_i is deployed, it is preconfigured with a non-migratable public key pair (e_{CH_i}, d_{CH_i}) and a hash chain C^{CH_i}. The seed value $c_0^{CH_i}$ of the hash chain is generated on CH_i using the TPM's physical random number generator and used by the CPU to perform the additional computations. CH_i is assumed to possess only one valid platform configuration, denoted as P_{CH_i}. After CH_i is powered up, a measurement about each component (BIOS, bootloader, operating system, applications) is performed, and the related values are stored in the corresponding PCR registers. Each value of the hash chain C^{CH_i} is sealed to this platform configuration P_{CH_i}: $\{c_0^{CH_i}\}_{P_{CH_i}}^{e_{CH_i}}, \ldots, \{c_n^{CH_i}\}_{P_{CH_i}}^{e_{CH_i}} = \texttt{Seal}(P_{CH_i}, e_{CH_i}, c_0^{CH_i}), \ldots, \texttt{Seal}(P_{CH_i}, e_{CH_i}, c_n^{CH_i})$.

Each CN_j which interacts with CH_i is configured with the last value $c_n^{CH_i}$ of the hash chain C^{CH_i}. Since the number of CHs is very small compared to the number of CHs, the CNs could be preprogrammed with the values of all CHs.

After deployment, the CNs can only keep the values for its CH and another certain number of CHs in their vicinity to save memory.

Attestation. CH_i and the associated CNs (denoted as CN_*) are loosely time synchronized. The time is divided into intervals I_λ, $\lambda = 1, \ldots, n$. At the beginning of each interval, CH_i sends a broadcast attestation message to the CNs. The attestation messages consist of the values of the hash chain released in reversed order of the generation and the identifier I_λ of the current interval. If the platform configuration of CH_i has not been modified, it can unseal the values of the hash chain C^{CH_i}. In the first interval I_1, CH_i unseals the hash value $c^{CH_i}_{n-1}$ and transmits it together with the interval identifier. In the second interval $c^{CH_i}_{n-2}$ is unsealed and transmitted and so on. CN_* check if the interval I_1 stated within the message matches their local interval counter I'_1 within a certain error range. If they match, CN_* verify whether $h(c^{CH_i}_{n-1}) = c^{CH_i}_n$. If the equation holds, CH_i is considered trustworthy and the value $c^{CH_i}_n$ is overwritten with the value $c^{CH_i}_{n-1}$. In the next interval CH_i releases $c^{CH_i}_{n-2}$ and so on, which are similarly checked. The protocol is shown in Figure 1 and repeated from $\lambda = 1$ to n.

Interval	Node(s)	Message	Action
I_λ	CH_i		$\texttt{Unseal}(P_{CH_i}, d_{CH_i}, \{c^{CH_i}_{n-\lambda}\}^{e_{CH_i}}_{P_{CH_i}}) = c^{CH_i}_{n-\lambda}$
I_λ	$CH_i \rightarrow CN_*$:	$c^{CH_i}_{n-\lambda}, I_\lambda$	
I_λ	CN_*		$I_\lambda \overset{?}{=} I'_\lambda$
I_λ	CN_*		if $h(c^{CH_i}_{n-\lambda}) \overset{?}{=} c^{CH_i}_{n-\lambda+1}$, state of CH_i is valid
I_λ	CN_*		overwrite $c_{n-\lambda+1}$ with $c_{n-\lambda}$
...	...	...	...

Fig. 1. Periodic Broadcast Attestation Protocol

Due to unreliable communication, a CN could miss some messages. Thus, CNs should not immediately declare a CH as being untrustworthy but wait for a certain threshold of time. If a CN receives messages again, it can resynchronize by applying the hashfunction multiple times.

5.2 Individual Attestation Protocol (IAP)

Using the IAP, a CN (or the sink) can individually verify the platform configuration of a CH. Alternatively a CN can send data to a CH and receive a confirmation that the data has been received correctly and that the CH is trustworthy. A CN needs only to perform symmetric operations and two short messages need to be exchanged. The messages are very small, because no long public key primitives, e.g., keys, signatures need to be transmitted. Since transmitting messages is the most cost intensive factor in WSNs [13], this is of particular interest, especially if the sink wants to verify the platform configuration of a CH. In this case, messages are transferred along several hops.

The protocol we propose is again divided in *initialization* phase and *attestation* phase. The initialization phase is performed only once after deployment of the sensor nodes while the attestation phase can be performed every time a CN (or the sink) wants to verify the platform configuration of a CH.

Initialization. Each CN_j establishes a shared, symmetric key K_{CN_j,CH_i} with its CH_i. Therefore, existing (non TPM-based) techniques, e.g., [14], might be used. However, we recommend using the key establishment protocol presented in [15], as it also assumes a hybrid WSN with TPM-equipped CHs and resource constraint CNs. This approach has the advantage that key generation within a TPM is inherently more secure than key generation on off-the-shelf embedded WSN platforms. As in [14], we assume that this short period of time to establish pairwise keys is secure and nodes cannot be compromised. The keys K_{CN_j,CH_i} are sealed on CH_i to its valid platform configuration P_{CH_i}. Thus, CH_i can access these keys only if it is in its valid state.

To enable the sink to perform the attestation with CH_i, a shared symmetric key K_{Sink,CH_i} is preconfigured on CH_i before deployment and sealed likewise.

Attestation. Figure 2 shows how CN_j can verify the platform configuration of CH_i. First, CN_j sends a challenge to CH_i. The challenge consists of an encrypted block containing a *Nonce* and the identifier ID_{CN_j} of CN_j, and additionally ID_{CN_j} in cleartext. K_{CN_j,CH_i} is used for encryption. After receiving the challenge, CH_i unseals K_{CN_j,CH_i} related to ID_{CN_j}. This is only possible if the platform configuration P_{CH_i} is valid. Using this key, CH_i decrypts the encrypted block and verifies if the decrypted identifier is equal to the identifier received in cleartext. If they match, CH_i knows that this message originates from CN_j, encrypts the *Nonce'* using K_{CN_j,CH_i}, and sends it back.[1] Otherwise, CH_i aborts. CH_i then deletes K_{CN_j,CH_i} from the RAM. CN_j decrypts the received response message and checks if the decrypted *Nonce"* matches the *Nonce* it has sent in the first step. If they match, CH_i is declared trustworthy and CN_j can send data to CH_i. This data is encrypted using K_{CN_j,CH_i}. The attestation of CH_i by the sink is performed analog, using the key K_{Sink,CH_i}.

Alternatively, data can be transmitted directly within the challenge. This might be preferable in scenarios where an immediate receipt of data is important or where CNs send data very infrequently to a CH. Therefore, the protocol is modified in steps 1 and 2b. Figure 3 shows the modifications. In step 1' CN_j sends the data to CH_i within the encrypted block. CH_i can only decrypt this message in step 2b' if its platform configuration is valid and access the data. All other steps remain the same as shown in figure 2. Thus, if CN_j receives the message in step 2e and the checks in steps 3a and 3b succeed, CN_j can be assured that CH_i has successfully received the data and is still trustworthy.

[1] However, the trustworthiness of CN_j cannot be assumed, because the node could be potentially compromised and the key is not protected by a TPM.

1. $CN_j \rightarrow CH_i : ID_{CN_j}, \{Nonce, ID_{CN_j}\}_{K_{CN_j,CH_i}}$
2a. CH_i $\texttt{Unseal}(P_{CH_i}, d_{CH_i}, \{K_{CN_j,CH_i}\}^{e_{CH_i}}_{P_{CH_i}}) = K_{CN_j,CH_i}$
2b. CH_i $D(\{Nonce, ID_{CN_j}\}_{K_{CN_j,CH_i}}, K_{CN_j,CH_i}) = (ID'_{CN_j}, Nonce')$
2c. CH_i check $ID'_{CN_j} \overset{?}{=} ID_{CN_j}$
2d. CH_i $E(\{Nonce', ID_{CH_i}\}, K_{CN_j,CH_i}) = \{Nonce', ID_{CH_i}\}_{K_{CN_j,CH_i}}$
2e. $CH_i \rightarrow CN_j : ID_{CH_i}, \{Nonce', ID_{CH_i}\}_{K_{CN_j,CH_i}}$
2f. CH_i delete K_{CN_j,CH_i} from RAM
3a. CN_j $D(\{Nonce', ID_{CH_i}\}_{K_{CN_j,CH_i}}, K_{CN_j,CH_i}) = (Nonce'', ID'_{CH_i})$
3b. CN_j if $Nonce'' \overset{?}{=} Nonce$, state of CH_i is valid

Fig. 2. Individual Attestation Protocol

1'. $CN_j \rightarrow CH_i : ID_{CN_j}, \{Nonce, ID_{CN_j}, data\}_{K_{CN_j,CH_i}}$
2b'. CH_i $D(\{Nonce, ID_{CN_j}, data\}_{K_{CN_j,CH_i}}, K_{CN_j,CH_i}) = (ID'_{CN_j}, Nonce', data)$

Fig. 3. Modified Individual Attestation Protocol

6 Analysis

In this section, we first discuss the security of the two proposed attestation protocols. Then we evaluate their performance.

6.1 Security Discussion

The goal of both protocols is that CNs can prove the trustworthiness of CHs. If an adversary compromises a CH, he cannot successfully deceive the CNs or the sink to perform insider attacks. We distinguish between two types of possible attacks: (1) attacking a CH directly, and (2) en-route attacks if the communication involves multiple hops. Due to the unreliable multihop communication, we can only prove the trustworthiness of CHs. But untrustworthiness could not be proven since either communication errors can result in modified attestation messages or malicious en-route nodes can modify forwarded messages to defame a CH. Therefore, a valid attestation makes no statement about the trustworthiness of a used route. In addition, an invalid attestation could be caused either by a compromised CH or by a compromised en-route CN.

Security of the PBAP. To compromise a CH and forge a trustworthy platform configuration, an adversary needs access to the hash chain. Therefore, he has to either perform the unseal command under a compromised platform configuration, or try to access the key used to seal the hash chain with physical attacks. As described in Section 2 the TPM is basically a smartcard and offers high security mechanisms for preventing unauthorized extraction of protected keys. This makes it extremely difficult for an adversary to retrieve the necessary keys to decrypt the sealed hash chain. Additionally, access to the sealed hash chain is only possible if the platform configuration has not been modified. This prevents the unauthorized extraction of the values of the hash chain in a compromised system environment. Even if an adversary could access the RAM of a sensor node, he can not retrieve other hash values, because for each attestation only the actual hash value is unsealed and loaded into the RAM.

However, our approach can not handle runtime attacks caused by buffer overflows, since we report the platform configuration measured in the initialization phase, i.e., when the software is first executed. Such attacks would result in a (malicious) modified system configuration, but the platform configuration stored in the PCRs is still the valid configuration.

If the attestation is performed between nodes which are multiple hops away, an adversary might also try to perform a man-in-the middle attack by compromising an en-route CN. The adversary can try to spoof, alter or replay attestation messages, or perform a selective forwarding attack [16]. Spoofing is not possible, because PBAP is not an authentication protocol. It gives an assertion about the trustworthiness of the specific CH and not which node has relayed the message. Altering attestation messages is possible and results in an unsuccessful attestation. To cope with that, a CN should possess an additional mechanism which enables the CN to reach its CH using a different communication path or change to a different CH. The CN can then use an alternative path and perform the IAP with the CH to make a clear statement, whether the route, or the node has been compromised. If the CH is compromised, a CN could, for example, switch to another CH where the communication paths and the new CH may not have been compromised. Replay attacks or an attack where an adversary first blocks the forwarding of legitimate hash values to collect them, then compromises a CH and finally releases these hash values are not possible, because hash values are only valid for a specific interval, which is validated by each CN. Since the PBAP is performed in cleartext an adversary can distinguish between attestation and data messages and therefore perform a selective forwarding attack by forwarding attestation messages, but blocking data messages. Such attacks are a general problem in WSNs and show that the PBAP is not resistant against all attacks in a multihop scenario with malicious en-route CNs.

Security of the IAP. The security of the IAP relies on the sealing of the symmetric keys to the valid platform configuration analogue to the sealing of the hash chain described above. Thus, an adversary compromising a CH cannot access the necessary keys to perform a successful attestation.

If the attestation messages are forwarded along multiple hops, an adversary can try to perform a man-in-the-middle attack. Since the IAP includes an authentication protocol, spoofing is not possible. A CH detects the modification of the first attestation message (see Figure 2) by an en-route adversary, since the included identifier does not match the identifier sent in cleartext. If the adversary alters the response sent to a CN, the latter cannot distinguish if either the attestation has failed or if the message has been altered by the adversary. Replay attacks are not possible, because a new Nonce is used in each message. Since attestation messages and data messages have the same form (identifier plus encrypted data block), an adversary cannot distinguish between them to perform a sophisticated selective forwarding attack. If the modified IAP is used, where data is sent in the first step, an adversary might be able to distinguish between this message and the response message (step 2e) because of the different lengths of the messages. To cope with that, the message sent in step 2e could be padded to the same length.

Thus, if an attestation fails, a CN should first try to perform a new attestation of the same CH using another communication path, if possible. If this is not possible or the attestation fails again, either the CH or a node on the communication path is compromised. The CN should then select a new CH, since messages sent to the old one might be susceptible to attacks.

Furthermore, in contrast to WSNs where CHs are not equipped with a TPM, a single compromise of a CH does not result in the compromise of all shared keys stored on this node. Even using the TPM in only a few sensor nodes results in a higher resiliency to node compromise.

6.2 Performance Analysis

Efficiency is crucial for security protocols for WSNs because of the scarce resources. Protocols should not introduce a high storage overhead and should not significantly increase energy consumption. Since we assume that CHs possess sufficient ressources, we perform our analysis only for the CNs. First, we analyse the additional storage requirements. Next, we estimate the additional energy consumption by evaluating the computational and communication overhead.

Storage Requirements. For the PBAP, a CN must store one hash value and the identifier for the corresponding CH. Depending on the network configuration, it might also store hash values (and identifiers) for other CHs in its vicinity. Let L_N, and L_H denote the length of a *node identifier* and a *hash value* respectively. Let the number of CHs for which a CN stores values be v. Thus, the storage requirements SR_{PBAP} for a CN are:

$$SR_{PBAP} = v * (L_N + L_H) \tag{2}$$

For example, suppose a CN stores values for 5 CHs. The length of each hash value is 64 bits and the length of a node identifier is 10 bits. This results in a total of 46.25 bytes.

For the IAP, a CN must store one symmetric key for each CH with which it wants to perform an attestation. Let this number be denoted by w and the length of a key denoted by L_K. Thus, the storage requirements SR_{IAP} for a CN are:

$$SR_{IAP} = w * L_K \tag{3}$$

For example, suppose a CN stores keys for 5 CHs. The length of each key is 64 bits. This results in a total of 40 bytes.

The Berkeley Mica2 Mote [8] offers 4KB of SRAM. Therefore, the storage requirements are suitable for current sensor nodes, even if both protocols are used in conjunction.

Energy Consumption. The PBAP requires a CN to receive one attestation message and to perform one hash computation at each time interval. An attestation message consists of a hash value and an identifier of the interval, e.g., a counter. Although computing hash values only marginally increases energy consumption [12], we consider the computational overhead, since a hash computation is performed in each time interval.

We use $e_1 = e_{1s} + e_{1r}$ to denote the energy consumed in sending and receiving one byte, and e_2 to denote the energy for one hash computation. In addition to the notation used above, let L_T denote the length needed for the interval identifier. The total number of intervals in the whole lifetime of the network is denoted with t. This results in an additional energy consumption:

$$E_{PBAP} = t * ((L_T + L_H) * e_{1r} + e_2) \tag{4}$$

For example, suppose the lifetime of the network is one year and broadcast messages are sent every 10 minutes. Therefore, a 16 bit counter is sufficient for numbering each interval. We use the results presented in [13] to quantify $e_{1s} = 16.25\ \mu J$ for sending, $e_{1r} = 12.5\ \mu J$ for receiving, and $e_1 = 28.75\ \mu J$ for sending and receiving one byte using Berkeley Mica2 Motes. The energy consumed for performing one hash computation using RC5 [17] block cipher is $e_2 = 15\ \mu J$. This results in a total energy consumption of 7358.4 mJ. The Mica2 Motes are powered with two 1.5 V AA batteries in series connection. We assume a total capacity of 2750 mAh using standard AA batteries which results in 29700 J. Thus, the ratio of energy consumed in one year by the PBAP is about 0.025% of the total available energy which is neglibly small.

The IAP requires a CN to generate and send a challenge[2], and the verification of the response (see Figure 2, steps 1, 2e, 3a and 3b). The challenge requires one Nonce generation, one encryption and one transmission, while the response verification requires the receipt of one message, one decryption and one comparison of two values. As in [12], the Nonce is generated using a Message Authentication Code (MAC) as pseudo-random number generator with a generator key $K^{rand}_{CN_j}$. The energy consumed therefor using RC5 for MAC generation is $e_2 = 15\ \mu J$. The encryption cost using RC5 are also 15 μJ. We neglect the energy cost for the comparison of two values since they are negligibly small. Thus, the additional energy consumption is:

$$E_{IAP} = 3 * e_2 + e_{1s} * (2 * L_N + L_H) + e_{1r} * L_H \tag{5}$$

[2] We do not consider the case where data is sent within the challenge, because we estimate only the additional overhead introduced by our protocol.

Assuming the values from above, this results in a total energy consumption on a CN for one individual attestation of about 315 μJ which is $1.06 * 10^{-4}$ % of the total available energy.

7 Related Work

In the context of attestation in WSNs, a number of software-based approaches have been presented [2,3,4] which rely on optimal program code and exact time measurements. These approaches enable software-based attestation by introducing an optimal program verification process that verifies the memory of a sensor node by calculating hash values of randomly selected memory regions. However, these approaches are not applicable in multihop WSNs, since they require, on the one hand, an authenticated communication channel between the verifier and the attestor, and on the other hand, rely on minimal time fluctuations (compare Section 3). In [18] a similar approach is presented which relies on code obfuscation techniques and time measurement. Proposed hardware-based approaches [10,6] are based on public-key cryptography and require extensive computational power, as well as the transmission of large messages, making these approaches not usable in WSNs. In [15] the advantages of using a TPM in hybrid WSNs are first identified. A framework for key establishment, distribution, and management is presented. The approach shows that a TPM can dramatically improve the security of WSNs. However, attestation techniques offered by the TPM are not considered.

8 Conclusions and Future Work

In this paper we argue that timing-based software attestation techniques are not applicable in multihop WSNs. We therefore introduce another approach which exploits the property of a hardware-based trust anchor to enable attestation in multihop WSNs. In this context, we present two attestation protocols for hybrid WSNs, where the network consists of resource constrained CNs and CHs with more resources equipped with a TPM chip that acts as a trust anchor. Both protocols allow CNs to verify whether the platform configuration of a CH is trustworthy or not, even if they are multiple hops away. The PBAP runs in fixed time intervals, allowing multiple nodes to verify the trustworthiness simultaneously, while the IAP enables a direct attestation. We shown that both the overhead for storage and the energy consumption are negligible.

We are currently working on the implementation of our proposed architecture using both hardware TPM and a TPM emulator. Furthermore, we are investigating how a CN should react if a multihop attestation, either based on a compromised node on the route or a compromised CH fails. To achieve this, an efficient algorithm must be developed that either selects a new route to the existing CH or chooses a new CH. Another part of our future work will be the choice of optimal parameters for CHs and CNs, taking the cost benefit aspects into account.

References

1. Akyildiz, I., Su, W., Sankarasubramaniam, Y., Cayirci, E.: A survey on sensor networks. IEEE Comm. Mag. 40(8), 102–114 (2002)
2. Seshadri, A., Perrig, A., Doorn, L.v., Khosla, P.: SWATT: SoftWare-based ATTestation for Embedded Devices. In: IEEE Symp. on Sec. and Priv., IEEE Computer Society Press, Los Alamitos (2004)
3. Seshadri, A., Luk, M., Shi, E., Perrig, A., Doorn, L.v., Khosla, P.: Pioneer: verifying code integrity and enforcing untampered code execution on legacy systems. In: SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles, Brighton, United Kingdom, pp. 1–16. ACM Press, New York (2005)
4. Seshadri, A., Luk, M., Perrig, A., van Doorn, L., Khosla, P.: SCUBA: Secure Code Update By Attestation in Sensor Networks. In: WiSe '06: Proceedings of the 5th ACM workshop on Wireless security, Los Angeles, California, ACM Press, New York (2006)
5. Group, T.C.: Trusted Platform Module (TPM) specifications, Technical report (2006) `https://www.trustedcomputinggroup.org/specs/TPM`
6. Stumpf, F., Tafreschi, O., Röder, P., Eckert, C.: A Robust Integrity Reporting Protocol for Remote Attestation. In: WATC'06. Proceedings of the Second Workshop on Advances in Trusted Computing (2006)
7. Shi, E., Perrig, A., Van Doorn, L.: BIND: A Fine-Grained Attestation Service for Secure Distributed Systems. In: SP '05. Proceedings of the 2005 IEEE Symposium on Security and Privacy, pp. 154–168. IEEE Computer Society Press, Los Alamitos (2005)
8. Crossbow Technology: Mica2 datasheet `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf`
9. Crossbow Technology: Stargate datasheet `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/Stargate_Datasheet.pdf`
10. Sailer, R., Zhang, X., Jaeger, T., Doorn, L.v.: Design and Implementation of a TCG-based Integrity Measurement Architecture. In: 13th USENIX Security Symposium, IBM T. J. Watson Research Center (August 2004)
11. Lamport, L.: Password authentication with insecure communication. Commun. ACM 24(11), 770–772 (1981)
12. Perrig, A., Szewczyk, R., Tygar, J.D., Wen, V., Culler, D.E.: SPINS: security protocols for sensor networks. Wirel. Netw. 8(5), 521–534 (2002)
13. Ye, F., Luo, H., Lu, S., Zhang, L.: Statistical en-route filtering of injected false data in sensor networks. In: Proceedings IEEE INFOCOM., IEEE Computer Society Press, Los Alamitos (2004)
14. Zhu, S., Setia, S., Jajodia, S.: LEAP: efficient security mechanisms for large-scale distributed sensor networks. In: CCS '03. Proceedings of the 10th ACM conference on Computer and communications security, ACM Press, New York (2003)
15. Ganeriwal, S., Ravi, S., Raghunathan, A.: Trusted platform based key establishment and management for sensor networks (Under review)
16. Karlof, C., Wagner, D.: Secure routing in wireless sensor networks: attacks and countermeasures. In: Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications, pp. 113–127. IEEE Computer Society Press, Los Alamitos (2003)
17. Rivest, R.L.: The RC5 Encryption Algorithm. In: Proceedings of the 1994 Leuven Workshop on Fast Software Encryption, pp. 86–96. Springer, Heidelberg (1995)
18. Shaneck, M., Mahadevan, K., Kher, V., Kim, Y.: Remote software-based attestation for wireless sensors. In: Molva, R., Tsudik, G., Westhoff, D. (eds.) ESAS 2005. LNCS, vol. 3813, Springer, Heidelberg (2005)