# Detection of Node Capture Attack in Wireless Sensor Networks

Sarita Agrawal*, Manik Lal Das*, and Javier Lopez†
*DA-IICT, Gandhinagar, India
†University of Malaga, Malaga, Spain

***Abstract*—Wireless sensor networks (WSNs) deployed in hostile environments for applications such as battle-field surveillance are vulnerable to various types of attacks, including node capture attack wherein an adversary physically captures, reprograms and redeploys a node in the network, hence gaining the advantages of an insider attacker. In this paper, we present a novel approach of program integrity verification (PIV) protocol to detect if a node is captured. The cluster head equipped with Trusted Platform Module (TPM) verifies by comparing the program memory content of the sensor node before and after capture. The proposed TPM-enabled PIV (TPIV) protocol uses dynamically computed hash based key and pseudo random function for detection of a captured node in the network. The security analysis of the TPIV protocol reveals that the probability of a node capture attack victim eluding the PIV and leaking the secret of any non-captured node is negligible. The proposed TPIV protocol can detect the captured node even in the presence of a strong adversary capable of putting additional memory to elude the PIV. With the results of analytical and experimental comparisons we show the performance improvement of TPIV protocol in terms of low communication, computation and storage overhead as compared to the related protocols for PIV in WSN.**

***Index Terms*—Wireless Sensor Network; Node Capture Attack; Trusted Platform Module; Program Integrity Verification.**

## I. Introduction

In most of the real-life applications such as battle-field surveillance and forest fire detection using wireless sensor networks (WSNs), a group of sensor nodes are densely and randomly deployed and are left unattended in hazardous conditions. The sensor nodes collaboratively monitor events such as movement of enemy troops in battle-field and communicate with each other over wireless channel. The sensor nodes can directly send the observations to a central trusted powerful entity, called base station, or through the intermediate nodes such as cluster heads. The wireless nature of the communication channel and the unattended deployment in difficult terrains leave the network vulnerable to various attacks including node capture. On the other hand, the inherent resource constraints of sensor nodes restrict using expensive security solutions for WSN. Although researchers have addressed various aspects of WSN security such as secure key management [2], secure localization and data aggregation, the problem of node capture attack is still a major concern in WSN. In a node capture attack, an attacker gets hold of a node physically, and then reprograms and redeploys the node. The attack severity depends on the time and resources available with the attacker. Various measures for increased resilience to node capture attack have been proposed in the literature through key management schemes; however, detection of node capture attack remains a challenging research problem. As the nodes in a WSN continuously interact with their neighboring nodes, an unusual absence resulting from physical capture of a node can be noticed with periodic monitoring. Various protocols [3], [4] have been proposed for detection of node capture by continuous monitoring. However, with the malicious neighbor collusion, the non-captured malicious nodes may intentionally report an absent node to be present in the network. As the absence of the node is not noticed within the defined threshold period, the adversary may reprogram and redeploy the captured node in the network for achieving its intended objective. Researchers have proposed protocols for code attestation and program integrity verification (PIV) [5], [6], [7], [8] for detecting such redeployed victim nodes. The case of software attestation protocol [5] requires optimal program code and accurate time synchronization between verifier and prover. On the other hand, soft tamper proofing protocol [6] is vulnerable to impersonation and replay attacks. In replay attack, an attacker intercepts a

message and re-transmits the same message at a later time. In impersonation attack, the adversary uses a malicious sensor node to create one or more fake identities [20]. Furthermore, the protocols, [5] and [6], need a centralized verifier. In [7], a compromised verifier can reveal the program code of all nodes; while in the case of utilization of specialized hardware (e.g. Trusted Platform Module (TPM) [9]) in each sensor node, such as in hardware based protocol [8], cost overhead occurs when large sensor networks are considered. In [**?**], a trusted local agent is used for calculation of the execution time interval and checksum computation between between two logical entities on prover that also increases communication overhead.

***Contributions of the paper.*** The node capture attack is critical to any WSN application, therefore, the need to efficiently and securely detect a captured node is an interesting research problem. In this paper, we present a protocol to detect the node capture attack in a clustered WSN using program integrity verification. In the proposed TPM-enabled program integrity verification (TPIV) protocol, each cluster head managing a group of nodes in the network is equipped with TPM capabilities. A cluster head serves as a TPM-enabled Verification Server (TVS) to check whether a redeployed node is a victim of node capture by verifying the integrity of the sensor node program. In a clustered network setup, wherein the only trusted entity is the base station, the verifiers themselves may be compromised. Therefore, an important aspect of the proposed TPIV protocol is that only an authenticated server can execute the PIV for a suspect node. In the TPIV protocol, a node that is asked to prove its program integrity is allowed to ensure that the verifier is authenticated. In case, the verifier is compromised, it may declare a valid node to be a victim of node capture attack and therefore, revoking it from the network. An unauthenticated verifier may unnecessarily engage the valid nodes in verification process to deplete their resources. The TPIV protocol works in the presence of an active adversary capable of adding memory to the node and ensures that a captured node does not reveal the secrets of other nodes. The performance improvement of the proposed TPIV protocol in terms of less computation, communication and storage overhead is evident from the experimental results.

The remainder of the paper is organized as follows: In section II, we analyze the existing protocols used to detect/prevent node capture attack in WSN. In section III, we present the proposed TPIV protocol. In section IV, we analyze the security and performance of the TPIV protocol. In section V, we show the experimental results of the TPIV. We conclude the work in section VI.

## II. Analysis of Related Work

A SoftWare-based ATTestation technique (SWATT) [5] proposes external verification of the code being executed on embedded devices. The SWATT algorithm solely executes through software means, requires an optimal program code and accurate time measurement. In addition, the verification procedure may get corrupted in case of the node compromise. Park *et al.* [6] proposed a soft tamper proofing (STP) protocol using randomized hash functions. In STP, each verification demands PIV server (PIVS) authentication by the base station, resulting in a risk of single point failure. In [7], this issue was addressed by using distributed authentication protocol for PIV (DAPP), wherein a PIVS first proves its authenticity with the help of a small group of other PIVSs for each verification. Although, DAPP avoids in engaging base station for verifier authentication, it requires entire program codes for all the nodes stored in each PIVS, with the risk that compromises of one PIVS gives an adversary access to program codes of all the sensor nodes. Moreover, DAPP uses bi-variate polynomial based pair-wise key sharing, where a node computes the keys with its own verifier and the verifier's selected authenticating PIVSs, which incurs storage and computational overhead. A node can compute pair-wise key using its polynomial share with any other node/verifier in the network, thus, by capturing the node, an adversary can communicate within the entire network. In 2010, Jin *et al.* proposed USAS (Unpredictable Software-based Attestation Solution) protocol [17] that deploys dynamic node attestation chains, wherein the base station is required to initiate the attestation through an initiator node. Hardware attestation using TPM [9] is proposed in [8]. A TPM protects the system from malicious activities and unauthorized changes. TPM has a shielded memory accessible only to that TPM, a set of secure Platform Configuration Registers (PCRs) storing integrity measurements for BIOS and/or application code prior to execution, and a public-private key-pair. With current platform configuration and the

public key, a TPM can `seal` any data block that can be later `unsealed` using the TPM's private key and TPM platform configuration at the time of unsealing, matching with the sealing time configuration. A simple abstraction of *TPM_Seal*() and *TPM_Unseal*() is defined as [16]:

*PSeal*(PCS, PK_TPM, DataTS)

*PUnSeal*(PCUS, SK_TPM, DataS)

Here, PCS is platform configuration at sealing time, PK_TPM is public key of TPM, DataTS is data to be sealed, PCUS is platform configuration at unsealing time, SK_TPM is private key of TPM and DataS is the sealed data. Fig. 1 illustrates the sealing and unsealing process.

The TRAP (TPM-enabled remote attestation protocol) discussed in [8] allows a node to verify the program integrity for any peer node. TRAP suggests all sensor nodes to be equipped with TPM that results in high network setup cost, especially for large WSNs. The need of node-to-base station interaction for each verification incurs communication overhead, therefore, it is infeasible in many WSN application scenarios. Moreover, the integrity of challenge and response messages is unprotected [8]. Yang and Yen proposed a memory attestation protocol in [**?**] that uses trusted local agents and requires a special hardware security model to be embedded in all the sensor nodes. The FlexiCast protocol [19] suggests authenticated broadcast and software attestation with the help of base station. Zhao discussed about a node capture attack [10]; however, that work analyzes the $q$-composite scheme for key pre-distribution that mitigates the node capture vulnerability in the neighbor discovery phase. In [11], the node replication attack (replicating and inserting duplicate nodes back into the network at selected places) detection is proposed that adversary can carry out after capturing the node. However, neither [10] nor [11] discusses about detection of node capture attacks.

## III. TPIV: The Proposed Protocol

In this section, we present the protocol TPIV for detection of node capture attack in which TPM-enabled verifiers can timely detect a victim node with the help of program integrity verification.

### A. Adversary Model

We consider the existence of an active adversary capable of capturing a node physically and stealing its secret information. The adversary can take part in the network operations by impersonating a valid node. The adversary is capable of reprogramming and redeploying the captured nodes in order to launch various insider attacks such as sybil attack (malicious sensor node creating multiple false identities) and selective forwarding (forwarding of the selected packets to travel through a malicious node depending on some criteria). The adversary is also capable of putting additional memory in the node. We further assume that the adversary can not remain present in the network all the time due to its deployment in the hostile terrains.

### B. System Model

We consider two types of nodes present in the network. A small set of nodes called cluster heads act as program integrity verification servers and the other nodes are normal sensor nodes with limited resources (Fig. 2). The sensor nodes are having separate user/data and program memory. A TPM chip is embedded in each verification server (we refer it as TPM-enabled Verification Server (TVS)). In comparison to the normal cluster nodes, TVSs are resource rich in terms of storage, communication and computation. Base station is a resourceful trustworthy central authority that is responsible for overall control and interaction with the external world. Prior to deployment, the base station securely loads the boot code, main application code and public functions in the program memory of each node. The free space in the program memory of a node is filled with random incompressible bit strings unique to that node. Any two nodes (say, $A$ and $B$) have random incompressible bit strings $S_A$ and $S_B$ ($S_A \neq S_B$, $\{S_A, S_B\} \in Z_p^*$ for some large prime $p \geq q$ ($q$ is a large prime defining the size of the secret)), respectively, in their free spaces. Due to the uniqueness of random bit strings, the overall program memory content of each node will be different (Fig. 3).

In this model, the cluster heads equipped with the TPM are decided prior to deployment. However, if a node is unable to communicate with the current cluster head, it is allowed to choose a new cluster head within its communication range. As the network considers mobile nodes and cluster heads, the nodes and/or cluster heads are also capable of moving to join any other cluster.

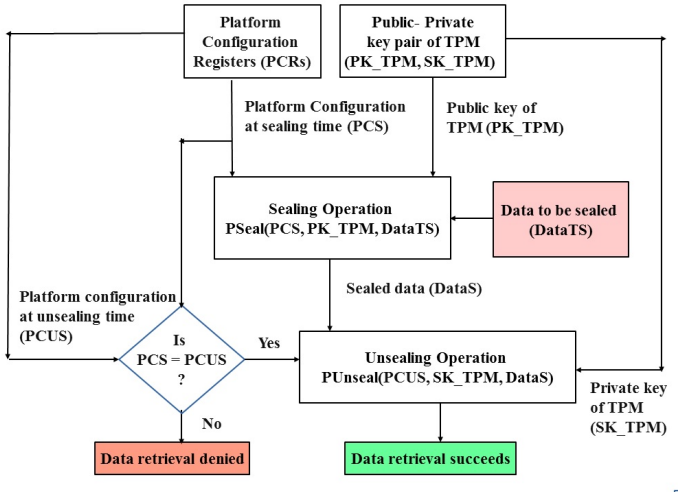



Fig. 1. Sealing and Unsealing with TPM

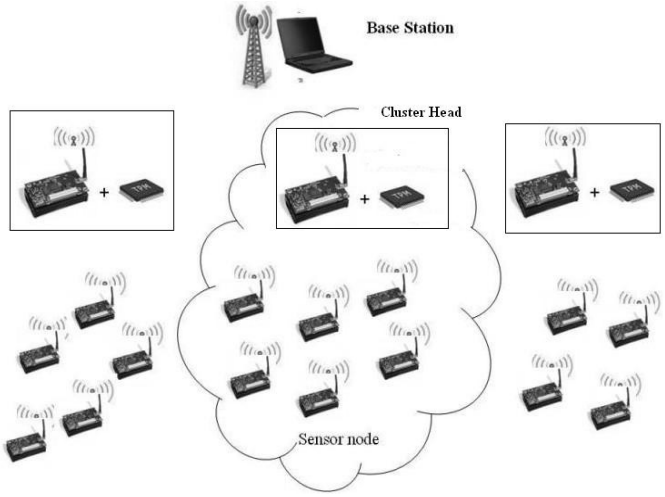



Fig. 2. Proposed Network Model

### C. Goals and Assumptions

The goal of the TPIV protocol is to detect node capture attack that ensures

- only an authorized verifier executes the program integrity verification for detecting a node capture suspect;
- a victim node can not elude the program integrity verification; and
- a captured node does not reveal the secret of other nodes.

Base station securely copies the free space contents of each node and one common copy of the remaining program memory content in each TVS. Using the initial platform configuration of TPM embedded in that TVS, the node program contents stored in a TVS are sealed. All nodes and TVSs are capable to compute a cryptographic hash function $h()$ and a cryptographic Pseudo Random Function $PRF_k()$ [13] using a key $k \in Z_q^*$ for a large prime $q$. We assume that the adversary does not know the program memory contents of any non-captured node. The hash of overall program memory content that includes a unique random bit string filled in the free space of a node serves as the initial secret between the sensor node and its cluster head. Post deployment, the nodes form a cluster by associating themselves with a cluster head nearest in their transmission range. When application code changes due to software update, free space is adjusted in accordance with the updated code size, assuming the code always leaves some reasonable

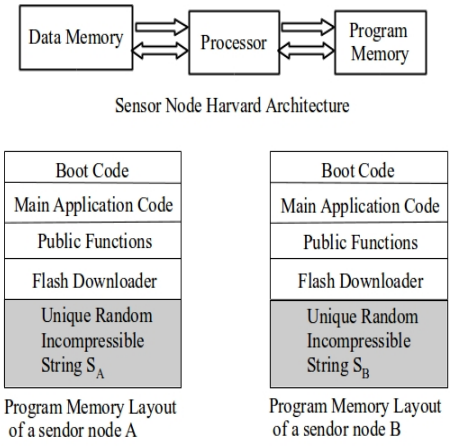



Fig. 3. Program Memory Layout

size of free space in the program memory. Any valid software update in the node is carried out by the base station through the cluster heads and thus, a cluster head is aware of such updates and can differentiate between a valid software updates and the unauthorized reprogramming of a node.

### D. Notations used

Table I gives the notations and symbols used in the proposed protocol.

| Notation | Description |
|---|---|
| TPM | Trusted Platform Module |
| PIV | Program Integrity Verification |
| TPIV | TPM-enabled PIV |
| TVS | TPM-enabled Verification Server |
| $ID_A$ | Identity of node $A$ |
| $CH_V$ | Identity of cluster head $V$ |
| $P_A$ | Program memory content of node $A$ |

TABLE I
NOTATIONS USED IN THE TPIV PROTOCOL

### E. Protocol Phases

The proposed TPIV protocol is executed with three phases: *System setup*, *Monitoring* and *Authentication and Code Verification*. In the system setup phase, the nodes and cluster heads are configured and deployed in the network. Monitoring phase is a continuous process in which a cluster head monitors the transmission of the nodes within its cluster to detect the unusual absence of a node from the network. During the Authentication and Code Verification phase, a verifying server and the suspect node mutually authenticate each other and then the verifier checks the program integrity of the suspect node.

*1) System Setup Phase:* Base station assigns a unique identity $ID_A \in Z^+$ to each node $A$ and $CH_V \in Z^+$ to each TVS $V$, prior to deployment. The TPM attached to each TVS $V$ has a unique non-migratable public-private key pair ($Pub_V$, $Pri_V$) that always residing in protected storage within TPM. At time $t$=0, TVS $V$ switches on and PCRs of associated TPM stores the initial platform configuration $PC_V^0$. Using $PC_V^0$ and $Pub_V$, the program memory content $P_A$ of node $A$ is sealed within the TPM associated with $V$ using $PSeal()$ function as

$$\mathcal{P}_{Asealed} = PSeal(PC_V^0, Pub_V, P_A)$$

This sealed content $\mathcal{P}_{Asealed}$ can be unsealed later at time $t$ using $PC_V^t$, the platform configuration of $V$ at time $t$ and $Pri_V$ (provided the configuration $PC_V^t = PC_V^0$) as $P_A = PUnseal(PC_V^t, Pri_V, \mathcal{P}_{Asealed})$. If $PC_V^t \neq PC_V^0$, then the unsealing fails and the sealed contents become inaccessible to TVS $V$.

*2) Monitoring Phase:* For each node $A$, the last transmission heard time $t_A^{last}$ is recorded by its cluster head. At time $t_A^{new} > t_A^{last}$, when next transmission is heard from $A$, if time interval between $t_A^{new}$ and $t_A^{last}$ is more than a set threshold time $T$ (i.e., $t_A^{new} - t_A^{last} \geq T$), the cluster head proceeds to perform Program Integrity Verification (PIV) for the suspect node $A$.

In the process of monitoring, the cluster head keeps a track of the communication from the cluster nodes. The set threshold is used to identify the unusual absence of a node from the network that may be due to the capture of the node. Our aim is to detect the node that becomes the victim of node capture attack and redeploy into the network after being reprogrammed. A predefined threshold time is measured based on the concept of statistical decision process such as Sequential Probability Ratio Test [21], [22]. If the absence time period of a sensor node is more than the threshold time, then the sensor node is suspected as the captured node. The threshold time to monitor the unusual absence is dynamically configured in accordance with the measured absence time duration for a sensor node. For instance, every sensor node measures the number of messages sent by its neighbors in a pre-defined time slot. With the monitoring process, if a neighbor does not send any message in the time slot, then it is believed that the neighbor has been captured.

*3) Authentication and Code Verification:* A TVS carries out the Program Integrity Verification (PIV) for a node suspected to be a victim of node capture attack. Before the node presents itself for such verification, it ensures the validity of the verifying TVS. The two-step protocol for Authentication and Code Verification is given in Fig. 4.

A TVS $V$, suspecting a node $A$, initiates the Authentication and Code Verification protocol. The TVS $V$, having the current platform configuration (PC) same as initial PC, retrieves the program memory content $P_A$ by unsealing the contents sealed into its TPM. TVS $V$ then computes the key shared with node $A$ as $K_A = h(P_A)$. TVS $V$ picks a nonce $N_V \in_R Z_q$ and sends the *PIV_Challenge* message as $CH_V$, $ID_A$, $N_V \oplus K_A$, $PRF_{K_A}(N_V, ID_A, CH_V)$ to node $A$ (Fig. 4: Step 1). On receiving *PIV_Challenge*, node $A$ extracts the nonce $N_V$ as $(N_V \oplus K_A) \oplus K_A$ and computes $PRF_{K_A}(N_V, ID_A, CH_V)$, where $K_A = h(P_A)$ and $P_A$ is the program memory of the sensor node $A$. The $P_A$ acts the secret key shared between TVS $V$ and sensor node $A$. If computed $PRF$ value matches with received $PRF$ value in *PIV_Challenge*, then the node $A$ is assured of the authenticity of the challenger TVS $V$. The sensor node $A$ then uses $N_V$ to compute the new key $K_{Anew} = h(P_{Acurr}, N_V, ID_A, CH_V)$ using its current program memory content $P_{Acurr}$. Then, the sensor node $A$ sends the *PIV_Response* $ID_A$, $CH_V$, $PRF_{K_{Anew}}(ID_A, CH_V)$, $PRF_{K_A}(N_V)$ to TVS $V$ (Fig. 4: Step 2) in reply to PIV_challenge. As the verification depends on the computation of new key $K_{Anew}$, the node $A$ sends $PRF_{K_A}(N_V)$ to assure the TVS $V$ of correct nonce received. When TVS $V$ receives *PIV_Response*, it first verifies the $PRF_{K_A}(N_V)$ and then computes new key as $K_V = h(P_A, N_V, ID_A, CH_V)$. TVS $V$ verifies the PRF value $PRF_{K_V}(ID_A, CH_V)$. If the value of $P_{Acurr}$ used to compute $K_{Anew}$ is same as the original value $P_A$ stored at TVS $V$, then the program code for node $A$ is unaltered and the PIV succeeds and TVS $V$ declares node $A$ to be authenticated and untempered. Otherwise, node $A$ is taken as captured and revocation of the captured node needs to be performed. At the end, node $A$ and TVS $V$ delete the nonce and keep the new secret $K_{Anew}$ (= $K_V$) for the next communication. When node $A$ leaves the cluster monitored by TVS $V$, node $A$ informs the move to TVS $V$ who informs this move to all the cluster heads and to all the nodes within its own cluster. The new TVS $W$, whose cluster is joined by node $A$ intimates this new join to old TVS $V$ who secretly shares the key $K_V$ with $W$ to communicate with node $A$. We note that a suspect node that is challenged to verify its program integrity is sent a nonce encrypted with the key $N_V \oplus K_A$ that is known only to the node and a valid verifier. A node responds to the verifier by computing the new key $K_{Anew}$ using the current program memory contents and the recent nonce shared by the verifier. If the verifier is compromised once the challenge is sent to the node, its platform configuration gets changed and that does not allow the verifier to access the node related data from its TPM and compute the new key.

## IV. Analysis

### A. Security Analysis

We analyze the proposed TPIV protocol against the security goals. The notations used in the security proofs are given in Table II.

| | |
|---|---|
| $Attacker(S)$ | Attacker has access to S (S may be a term/name/ variable/channel) |
| $W \wedge V$ | Logical AND operation on $W$ and $V$ |
| $W \vee V$ | Logical OR operation on $W$ and $V$ |

TABLE II
NOTATIONS USED IN THE SECURITY PROOFS

*Statement 1:* If $h(.)$ is a cryptographic hash function with collusion resistance defined as $h : \{0,1\}^m \rightarrow \{0,1\}^k$, then the probability of hash collision [15], i.e., finding a $x^{'} \neq x$, such that $h(x) = h(x^{'})$ for a given a $x$ and $h(x)$, is $Pr[(h(x) = h(x^{'}) \ (x^{'} \neq x))] = 1 - (1 - \frac{1}{2^k})^{m-1}$

*Statement 2:* If $s_1$ is a binary string of length $l$, then the maximum probability $p_{s_1 s_2}$ of finding another string $s_2$ having hamming distance at least one from $s_1$ is $\frac{l}{2^l}$.

The probability of selecting a bit string which has 0 hamming distance from a given bit string is $\frac{1}{2^l}$. Now, for a bit string to be at least one hamming distance away from another bit string, there would be at least one bit different. As the string length is $l$, there are $l$ such possible positions. Therefore, there are total $l$ strings which can have one bit different from a given bit string. Out of total $2^l$ possibilities of an $l$ size string, the probability of

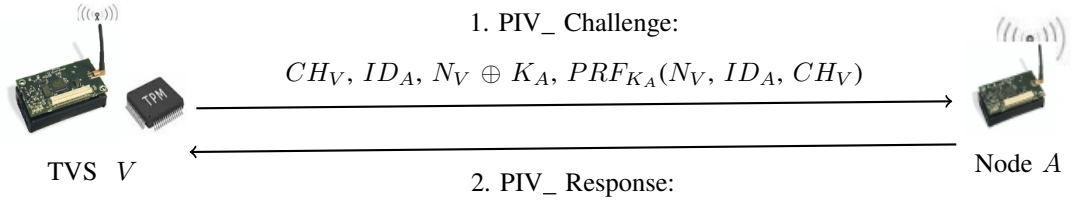



Fig. 4. Authentication and Code Verification protocol

finding another bit string having hamming distance at least one from the given bit string is $\frac{l}{2^l}$.

*Definition 1:* If $p_{succ}$ is the probability of a node successfully receiving a message from another node within its communication range, the probability of a verifier successfully receiving the response from challenged node is given as

$$Pr[TVS\ V \xleftarrow{PIV\_Response} Node\ A] = \begin{cases} p_{succ}, & \text{if TVS is authorized verifier} \\ 0, & \text{otherwise} \end{cases}$$

Here, $p_{succ} = \sum_{j=0}^{e^c} \binom{k}{j}.(1-e)^{k-j}.e^j$, on a channel with bit error rate $e$, for $k$ out of maximum $e^c$ number of bits encoded.

*Definition 2:* EludePIV($P_A$, $P_{Anew}$, $CH_V$, $ID_A$, $N_V$) determines if a captured reprogrammed node eludes the program integrity verification. Here, $P_A$ and $P_{Anew}$ are original and reprogrammed memory content of node $A$, respectively, $CH_V$ and $ID_A$ are the unique identities of TVS $V$ and node $A$, respectively and $N_V$ is the nonce received by node $A$ from server $V$. The function returns 0, if $h(P_A, N_V, CH_V, ID_A) \neq h(P_{Anew}, N_V, CH_V, ID_A)$; else, it returns 1.

***Theorem 1.*** The $Pr$[EludePIV($P_A$, $P_{Anew}$, $CH_V$, $Id_A$, $N_V$)] = TRUE is (1 - $(1 - \frac{1}{2^k})^{m-1}$) for $m$ = $|P_A \| N_V \| CH_V \| Id_A|$ and $k$ = $|h(P_A, N_V, CH_V, ID_A)|$.

***Proof.*** We prove that a victim of node capture attack can not elude the PIV. After capturing node $A$, the attacker had access to secret $K_A$ (i.e., $h(P_A)$). Since the attacker has access to the message $N_V \oplus K_A$ from public channel and has access to key $K_A$, attacker has access to $N_V$ as well. When the attacker reprograms this node, the program memory $P_A$ changes to $P_{Anew}$. As the free space in the memory is filled in by an incompressible random bit string, an adversary can not insert code pieces while keeping the original program intact [12]. When the attacker puts additional memory

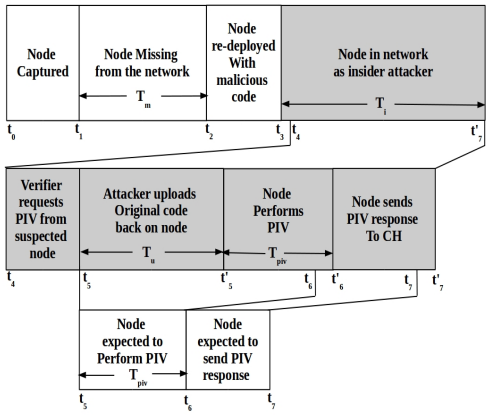



Fig. 5. Attacker Time Line

and keeps the original program code in it, at any point of time, the attacker has either $P_A$ or $P_{Anew}$ as program memory. Therefore,

| | |
|---|---|
| Attacker($K_A$) | ... (1.1) |
| Attacker($N_V$) | ... (1.2) |
| Attacker($P_A$) $\wedge$ Not Attacker($Pnew_A$) | ... (1.3) |
| Attacker($Pnew_A$) $\wedge$ Not Attacker($P_A$) | ... (1.4) |

To elude the verification, the victim node must swap out the malicious code $P_{Anew}$, and swap in the original code $P_A$ in the program memory before starting PIV protocol. Let us go through the time line for an attacker carrying out the node capture attack (refer to Fig. 5).

As per attack time line (Fig. 5), node $A$ is captured at time $t_1$ and redeployed with malicious code at time $t_3$. Since the node was missing between time $t_1$ and $t_2$, soon after as the node is available in the network, the verifier requests the PIV from node $A$. In order to hide the malicious code, the node $A$ starts uploading the original program code $P_A$ into the memory. The node

then starts the PIV execution at time $t'_5$ (= $t_5$ + $T_u$) instead of $t_5$ as expected by TVS. The node is expected to finish the PIV by time $t_6$ and the response should have reached back to the TVS by time $t_7$. However, due to the delay at the previous step, the victim node finishes the PIV at time $t'_6$ and therefore, starts sending the PIV_Response message at time $t'_6$ (= $t'_5$ + $T_{piv}$). As a result, TVS receives the delayed response at $t'_7$ ($> t_7$). Although the time synchronization in TPIV protocol is not crucial, the implementation (with ATmega328 processor using Arduino Duemilanove controller board and ArduinoISP programmer) of TPIV protocol at node end shows that to upload 9.4 KB of code associated only with PIV protocol takes about 11.27 seconds ($T_u$) which itself is significantly more than the protocol execution time ($T_{piv}$) measured as 5.86 seconds. Thus, the victim does not have enough time to use the original code to elude the PIV while having malicious code to carry out insider attacks otherwise.

Another possibility is the victim node $A$ eluding the PIV even after reprogramming i.e., holding $P_{Anew}$ as its program memory and, $A$ is able to send a valid PIV_Response to TVS $V$. Let us track back the protocol execution:

Attacker(valid PIV_Response)
- $\Rightarrow$ Attacker($PRF_{K_{Anew}}(ID_A, CH_V)$) $\wedge$ Attacker($PRF_{K_A}(N_V)$)
- $\Rightarrow$ Attacker($K_{Anew}$) $\wedge$ Attacker($K_A$) $\wedge$ Attacker($N_V$)
- $\Rightarrow$ Attacker($K_{Anew}$) $\wedge$ TRUE $\wedge$ TRUE [using assertions 1.1 and 1.2 respectively]
- $\Rightarrow$ Attacker($h(P_{Anew}, N_V, ID_A, CH_V) = h(P_A, N_V, ID_A, CH_V)$) $\vee$ (Attacker($P_{Anew}$) $\wedge$ Attacker($P_A$))
- $\Rightarrow$ TRUE (with $Pr$(1 - $(1-\frac{1}{2^k})^{m-1}$) $\vee$ FALSE [using Statement 1 and assertions 1.3/1.4]
- $\Rightarrow Pr$[EludePIV($P_A, P_{Anew}, CH_V, Id_A, N_V$)] = TRUE is (1 - $(1-\frac{1}{2^k})^{m-1}$) [by Definition 2]

The proof explains that if the attacker is able to send a valid PIV_Response that means it is able to compute both $PRF_{K_{Anew}}$ and $PRF_{K_A}$. This implies that the attacker got hold of new key $K_{Anew}$, old key $K_A$ and the nonce $N_V$. From assertions 1.1 and 1.2, we know that attacker does have $K_A$ and $N_V$; however, getting $K_{Anew}$ indicates that either attacker got $h(P_{Anew}, N_V, ID_A, CH_V)$ which is equal to $h(P_A, N_V, ID_A, CH_V)$ or attacker has both $P_{Anew}$ and $P_A$ at the same time in the node's program memory. Assertion 1.3 and 1.4 denies the possibility of later and the former possibility exists with the hash collision (1 - $(1-\frac{1}{2^k})^{m-1}$) $\square$

***Theorem 2.*** The probability of an unauthorized verifier executing the PIV for detecting a node capture attack is $\epsilon$, where $\epsilon$ is the hash-collision probability (ref. Statement 1).

***Proof.*** Consider a TVS $V$ compromised by an attacker. Being TPM enabled, any change in TVS $V$ program changes its platform configuration. The program code $P_A$ for the node $A$ can be unsealed from TPM of the verifier only when the current and initial platform configurations match. Let us consider the scenario that the TVS $V$ sends the 'PIV_challenge' to node $A$ in order to execute PIV. Therefore,

| | |
|---|---|
| Not Attacker($PC_V^0$) | ... (2.1) |
| Attacker($N_V$) | ... (2.2) |
| (as attacker sending 'PIV_challenge' message) | |

If the verifier is successful in executing the PIV for node $A$, the verifier must receive the 'PIV_Response' from node as reply to 'PIV_challenge'. Let us trace back the protocol execution: Attacker(PIV_Response)
- $\Rightarrow$ Attacker($PRF_{K_{Anew}}(ID_A, CH_V)$) $\wedge$ Attacker($PRF_{K_A}(N_V)$)
- $\Rightarrow$ Attacker($K_A$) $\wedge$ Attacker($N_V$)
- $\Rightarrow$ Attacker($K_A$) $\wedge$ TRUE (using assertion 2.2)
- $\Rightarrow$ Attacker($P_A$) (since $K_A$ = $h(P_A)$) $\vee$ Attacker($h'$ (= $h(P_A)$))
- $\Rightarrow$ Attacker($PC_V^t$(=$PC_V^0$)) ($P_A$ can be unsealed from TPM only with same current and initial configuration ) $\vee$ TRUE (with $Pr$(1- $(1-\frac{1}{2^k})^{m-1}$) (using Statement 1)
- $\Rightarrow$ FALSE [using assertion 2.1] $\vee$ TRUE (with $Pr$(1 - $(1-\frac{1}{2^k})^{m-1}$)
- $\Rightarrow$ TRUE (with $Pr$(1 - $(1-\frac{1}{2^k})^{m-1}$)

A compromised TVS receiving the response of PIV_challenge implies that the TVS was able to send a valid challenge. This in turn requires that the compromised TVS has either the same platform configuration as at the time of sealing or it could find out a hash value equal to $h(P_A)$. If TVS is tampered with, the platform configuration can not remain same. Therefore,

the only possibility is that the hash collision has occurred. Thus, even if an unauthorized verifier receives a message from a node with probability $p_{succ}$, the probability of the unauthorized verifier receiving 'PIV_Response' from challenged node, i.e., the probability of unauthorized verifier executing PIV for a node is equivalent to the hash-collision probability. □

The protocol does not allow an attacker to keep the hash of the program memory content pre-computed to be used for verification, as the nonce supplied by TVS has to be used for each verification round. For both theorems 2 and 3, with our implementation with $m$ = $2^{18}$ bits (for program memory size of 32 KB) and $k$ = 256 bits (secret size), the probability is (1-$(1-\frac{1}{2^{256}})^{2^{18}-1}$) $\approx$ 0.

***Theorem 3.*** Given the capture of a node $A$, the protocol leaks the secret of a non-captured node with probability $Pr$[Attacker($K_B$)/$K_A$] = $\frac{l}{2^l}$, where $l$ = log $p$ bits is size of random bit string.

***Proof.*** From the system model, the program memory contents of any two nodes differ by the difference in the unique random bit strings in their respective free spaces. When the adversary captures node $A$, he knows the entire program memory content $P_A$. Therefore,

| Attacker($P_A$ ($\neq P_B$)) ... (3.1) |
|---|

Let us claim that capture of node $A$ reveals the secret $K_B$ of node $B$. Then,

Attacker($K_B$)
$\Rightarrow$ Attacker($P_B$ = ($P_A$)) $\vee$ Attacker($P_B$)
$\Rightarrow$ Attacker($P_A$ = ($P_B$)) $\vee$ False
[using assertion 3.1]

The attacker can have access to program contents $P_B$ for a non-captured node $B$, if the contents of the node is similar to the contents of a captured node $A$. The probability of $P_B$ matching with $P_A$ even with one bit difference is = ($P_B$) is $\frac{l}{2^l}$ (by Definition 1). Thus, the capture of a node $A$ reveals the secret of any other node $B$ with probability $\frac{l}{2^l}$. Since $l$ = log $p$ ($\geq$ log $q$), with our implementation, even if we consider loq $p$ = loq $q$ = 256 bits, then $Pr$[Attacker($K_B$)/$K_A$] = $\frac{256}{2^{256}} \approx 0$. □

The proposed protocol provides security against replay and impersonation attacks [12]. Table III provides the comparison of the security features of the proposed TPIV protocol with some related protocols.

### B. Performance Analysis

Table IV captures the performance of the proposed TPIV protocol in terms of storage, computation and communication overhead as compared with existing protocols. For computation overhead, we only consider the time incurred in three main categories of operations namely, public key, symmetric key and heavy TPM operations such as TPM_Unseal and TPM_Sign, ignoring the light weight operations such as XOR and TPM_Read. In the performance comparison table below IV, timings required to perform a public key operation, symmetric key operation and TPM Unseal/Sign operations are depicted as $T_p$, $T_s$ and $T_t$ respectively. Size of secret is given by log $q$ for a large prime number $q$. For DAPP protocol [7], $k$ is the degree of polynomial used for generating key, $s$ is total PIVSs in network and $N_{auth}$ is number of authentication tickets needed by a PIVS.

With this protocol, we achieved an overall performance and security improvement with the TPM enabled verifiers as compared to the software based scheme such as [7]. At the same time, we saved the cost to equip all the nodes with the TPM chip as in [8].

## V. Experimental Results

The performance of the existing software based DAPP [7] and TPIV protocol are further compared using experimental and simulation results. The network is simulated with Castalia simulator [14] for a field of size 100 by 100 with uniformly distributed varied number of nodes communicating with 25 meters of radio range.

We simulated the TPIV protocol with varied number of TVSs against a set of nodes. From Fig. 6, it can be seen that when the number of TVSs are increased from 5 to 10, the percentage increase of nodes served is high, however, when we increase the number of servers beyond 10, then either the percentage increase is very low or it decreases. We observed that as the number of servers increase in the network, the messages floating in the network increase and therefore, for total 100 nodes in the network, when the number of servers is increased to 25, the packet drop due to collision was high and therefore, the number of nodes served during a round started decreasing. With 10 TVSs, we

| Features ⇓ / Scheme ⇒ | SWATT [5] | STP [6] | DAPP [7] | TRAP [8] | TPIV |
|---|---|---|---|---|---|
| no accurate time synchronization | × | × | ✓ | ✓ | ✓ |
| challenge-response message integrity protected | ✓ | × | ✓ | × | ✓ |
| base station only involved prior to deployment | × | ✓ | ✓ | × | ✓ |
| mutual authentication between verifier-prover | ✓ | × | ✓ | ✓ | ✓ |
| compromised verifier can not execute verification | ✓ | × | × | ✓ | ✓ |
| captured node does not reveal valid node secrets | ✓ | ✓ | * | ✓ | ✓ |
| protected against node memory addition | ✓ | ✓ | × | ✓ | ✓ |

*$k$- collusion resistant for k-degree polynomial used for key

TABLE III
COMPARISON OF THE SECURITY FEATURES

| Features ⇒ / Protocol ⇓ | Prover or Verifier | Storage overhead (bits) | Computation overhead | Communication overhead (bits) | |
|---|---|---|---|---|---|
| | | | | Transmit | Receive |
| DAPP [7] | Verifier | ($k$+$s$) log $q$ | ($N_{auth}$ + $s$ +($s$ + 1)) $T_p$ | ((3 $N_{auth}$ + 8) $T_s$ + 7) log $q$ | (4 $N_{auth}$ + 8) log $q$ |
| | Prover | ($k$+$N_{auth}$ +2)log $q$ | ($N_{auth}$+7) $T_s$ + ($N_{auth}$ + 1) $T_p$ | 10 log $q$ | (2 $N_{auth}$ + 8) log $q$ |
| TRAP [8] | Verifier | 3 log $q$ | $T_t$+ 3 $T_s$ + $T_p$ | 3 log $q$ | 3 log $q$ |
| | Prover | log $q$ | 2 $T_t$ + 2 $T_s$ | 2 log $q$ | log $q$ |
| TPIV protocol | Verifier | log $q$ | $T_t$ + 4 $T_s$ | 4 log $q$ | 4 log $q$ |
| | Prover | log $q$ | 5 $T_s$ | 4 log $q$ | 4 log $q$ |

TABLE IV
PERFORMANCE COMPARISON

are able to achieve the optimum performance. Moreover, even with a network of 500 nodes, more than 20% nodes are served at a time with the help of 10 TVSs. In a practical scenario, this suffices to deal with the captured nodes effectively.

Fig. 7(a) and 7(b) show the significant improvement in the energy consumption and the average communication latency with the TPIV protocol. The reason for this is, in DAPP protocol, four rounds of to and from communication between the node and the server are required, whereas, TPIV protocol needs one round of communication. The sleep schedule of nodes affects the latency since TMAC as MAC protocol is used for simulation. For large number of nodes (say 1000), almost all the nodes remain awake for most of the time, as they are continuously getting signals from the neighboring nodes. In the TPIV protocol, an awake node can immediately respond back to the server challenge, while in DAPP, node has to wait for the

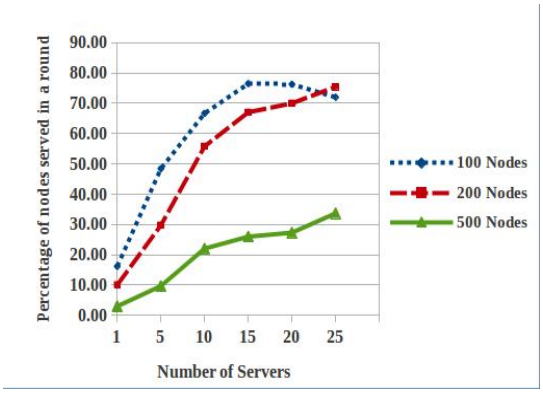


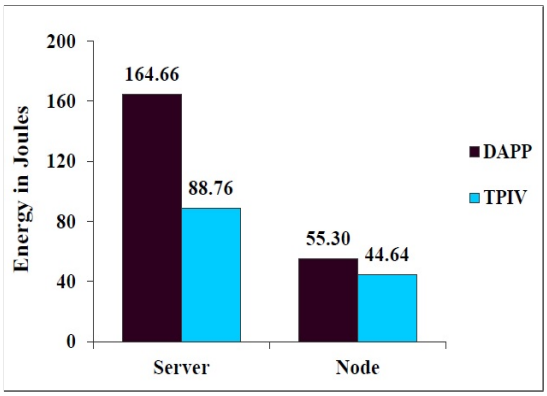

Fig. 6. TVS Optimization

server to collect authentication tickets from other servers.

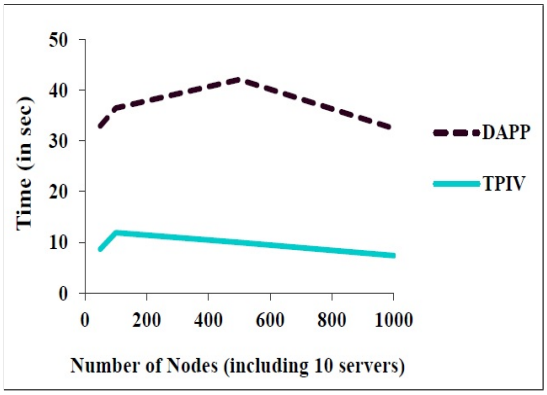
We implemented the protocols with ATmega328 processor using Arduino Duemilanove controller board and ArduinoISP programmer. The experimental results show that the computation cost incurred in executing the PIV with DAPP is significantly less than with TPIV (refer to Fig. 7(c)) [7]. The reason is that, in DAPP a node as well as the verifying server has to compute polynomial based keys for all authentication ticket providing TIVSs. Furthermore, to verify the authentication tickets, the MAC needs to be computed. In TPIV, a node performs only an un-keyed hash and 3 PRF operations.

## VI. Conclusion

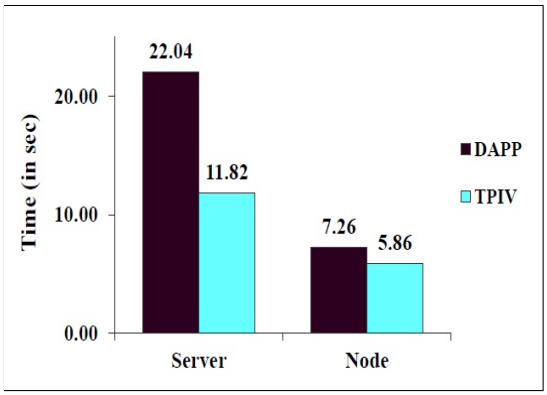
We have proposed a TPM-enabled Program Integrity Verification (TPIV) protocol to detect the node capture attack in a distributed WSN. The proposed TPIV protocol relies on the strength of cryptographic hash function and it is capable of securely and efficiently detect the node capture attack in the presence of an active adversary capable of putting additional memory in the node when captured. TPIV ensures that only an authorized verifier can execute the verification. Through experimental results it is proved that the protocol does not allow a victim node to elude the verification process. Moreover, the protocol prevents a captured node from revealing the secrets of other nodes. With TPM-enabled verifier sealing the program code of nodes, the protocol does not reveal node program code on verifier compromise. As evident from the performance analysis and simulation results, in comparison to the pure software based protocols, TPIV provides additional security with significant reduction in communication, computation and storage overhead on the nodes.




(a) Energy Consumption




(b) Communication Latency




(c) Computation Cost

Fig. 7. Performance comparison of TPIV and DAPP [7]

## References

[1] W. Ding, B. Laha and S. Yenduri. First Stage Detection of Compromised Nodes in Sensor Networks. In the proceedings of Sensors Applications Symposium, pp. 20-24, 2009.

[2] S. Agrawal, M. L. Das, R. Roman, A. Mathuria and J. Lopez. A Novel Key Update Protocol in Mobile Sensor Networks. In the roceedings of 8th International Conference on Information Systems Security, Springer LNCS Vol. 7671:194-207, 2012.

[3] M. Conti, R. Pietro, L. Mancini and A. Mei. Mobility and Cooperation to Thwart Node Capture Attacks in MANETs. In EURASIP Journal on Wireless Communications and Networking, Vol. 2009(8):1-8, 2009.

[4] M. Conti. Capture Detection. Book Chapter in Secure Wireless Sensor Networks: Threats and Solutions, Springer ISBN 978-1-4939-3460-7, pp. 53-73, 2016.

[5] A. Seshadri, A. Perrig, L. Doorn and P. Khosla. SWATT: SoftWare-based ATTestation for Embedded Devices. In the proceedings of the IEEE Symposium on Security and Privacy, pp. 272-282, 2004.

[6] T. Park and K.Shin. Soft-Tamper-Proofing via Program Integrity Verification in Wireless Sensor Networks. In IEEE Transactions on Mobile Computing, Vol. 4(3): 297-309, 2005.

[7] K. Chang and K. Shin. Distributed Authentication of

Program Integrity Verification in Wireless Sensor Networks. In ACM Transactions on Information and Systems Security, Vol. 11(3): 14:1-14:35, 2008.

[8] H. Tan, W. Hu and S. Jha. A TPM-enabled Remote Attestation Protocol (TRAP) in Wireless Sensor Networks. In the proceedings of the 6th ACM workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks, pp. 9-16, 2011.

[9] Trusted Computing Group. https://www.trustedcomputinggroup.org/ (retrieved 24 Jan 2017)

[10] J. Zhao. On Resilience and Connectivity of Secure Wireless Sensor Networks under Node Capture Attacks. In IEEE Transactions on Information Forensics and Security, Vol. 99: 1-1, 2016.

[11] A. Mishra and A. Turuk. A Comparative Analysis of Node Replica Detection Schemes in Wireless Sensor Networks. In Journal of Network and Computer Applications, Vol. 61: 21-32, 2016.

[12] S. Agrawal, M. L. Das, A. Mathuria and S. Srivastava. Program Integrity Verification for Detecting Node Capture Attack in Wireless Sensor Network. In the proceedings of International Conference on Information Systems Security, Springer LNCS Vol. 9478: 419-440, 2015.

[13] A. Menezes, P. Van. Handbook of Applied Cryptography, CRC Press, 1996.

[14] A. Boulis. Castalia-A Simulator for Wireless Sensor Networks and Body Area Networks-User's Manual, Version 3.2, 2011

[15] M. Peyraviana and A. Kshemkalyanib. On Probabilities of Hash Value Matches. In Computers and Security, Vol. 17(2): 171–174, 1998.

[16] C. Krauß, F. Stumpf and C. Eckert. Detecting Node Compromise in Hybrid Wireless Sensor Networks using Attestation Techniques. In Journal of Security and Privacy in Ad-hoc and Sensor Networks, pp. 203–217, 2007.

[17] X. Jin, P. Putthapipat, D. Pan, N. Pissinou, and S. K. Makki. Unpredictable Software-based Attestation Solution for Node Compromise Detection in Mobile WSN. In the proceedings of IEEE Globecom Workshop on Advances in Communications and Networks, pp. 2059–2064, 2010.

[18] P. Yang and S. Yen. Memory Attestation of Wireless Sensor Nodes by Trusted Local Agents. In the Proceedings of IEEE Trustcom/BigDataSE/ISPA, pp. 82–89, 2015.

[19] J. Lee, L. Kim and T. Kwon. FlexiCast: Energy-Efficient Software Integrity Checks to Build Secure Industrial Wireless Active Sensor Networks. In IEEE Transactions on industrial informatics, Vol. 12(1): 6–14 , 2016.

[20] J. Lopez, R. Roman, and C. Alcaraz. Analysis of Security Threats, Requirements, Technologies and Standards in Wireless Sensor Networks. In the Tutorial Lectures on Foundations of Security Analysis and Design V, Springer LNCS Vol. 5705: 289–338, 2009.

[21] J. Ho. Distributed Detection of Node Capture Attacks in Wireless Sensor Networks. In Smart Wireless Sensor Networks, pp. 345–360, 2010.

[22] A. Wald. Sequential Analysis. In Dover Publications, 2013.