COMPUTERS & SECURITY 81 (2019) 78–93

Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

# MTRA: Multi-Tier randomized remote attestation in IoT networks

*Hailun Tan*[a,*], *Gene Tsudik*[b], *Sanjay Jha*[a]

[a] *School of Computer Science and Engineering, the University of New South Wales, Australia*
[b] *Computer Science Department, University of California, Irvine, United States*

ARTICLE INFO

*Article history:*
Received 8 July 2018
Revised 15 September 2018
Accepted 19 October 2018
Available online 15 November 2018

*Keywords:*
Remote attestation
Internet of Things
Network security
Embedded systems
Network attack

ABSTRACT

Large numbers of Internet of Things (IoT) devices are increasingly deployed in many aspects of modern life. Given their limited resources and computational power, verifying program integrity in such devices is a challenging issue. In this paper, we design MTRA, a Multiple-Tier Remote Attestation protocol, by exploiting differences in resources and computational power among various types of networked IoT devices. More powerful devices equipped with a Trusted Platform Module (TPM) are verified through trusted hardware while others are verified through software-based attestation. In addition, a randomized memory region for attestation is used in MTRA to increase the entropy of the attestation responses. MTRA is a flexible means of program integrity verification for heterogeneous IoT devices.



A wide range of applications makes the Internet of Things (IoT) an attractive new paradigm. Various types of wireless devices, such as household appliances and office or automation devices, can connect with each other. The majority of the smart devices in IoT are resource-constrained and they have also processing and communication capabilities as well as they possess a locatable Internet protocol address (IP address, Challa et al., 2017) . The smart devices can be remotely accessed and controlled using existing network infrastructure which allows a direct integration of computing systems with the phisical environment. This facility further reduces human involvement, and also improves accuracy and efficiency that result in economic benefit . The smart devices in IoT facilitate the day-to-day life of people. In critical IoT settings such as defense, healthcare, and industrial process control, there is a clear need to ensure that some critical devices, such as door locks and vehicle smart control systems, cannot be maliciously manipulated from Internet. Fig. 1 shows that IoT networks is composed by various types of networks devices, which have different capabilities and responsibilities. Therefore, the security protocol design in IoT should also accommodate such differences accordingly.

In this paper, we propose MTRA, a Multiple-Tier Remote Attestation protocol, for IoT networks. Since IoT devices tend to be heterogeneous in terms of computational and communication capabilities, some are equipped with additional hardware, Trusted Platform Modules (TPMs), to assist with remote attestation. Simpler, smaller and cheaper devices might lack any such hardware. Accordingly, different attestation techniques are appropriate for different types of IoT devices.

The rest of the paper is organized as follows. Section 1 surveys the literature in remote attestations for wireless networks. Section 2 outlines the system and threat models on which this paper is based. Section 3 details MTRA to verify the code image of the devices. Section 4 discusses the poten-

* Corresponding author.
*E-mail addresses:* hailun.tan@unsw.edu.au, thailun@cse.unsw.edu.au (H. Tan), gts@ics.uci.edu (G. Tsudik), sanjay.jha@unsw.edu.au (S. Jha).
https://doi.org/10.1016/j.cose.2018.10.008
0167-4048/

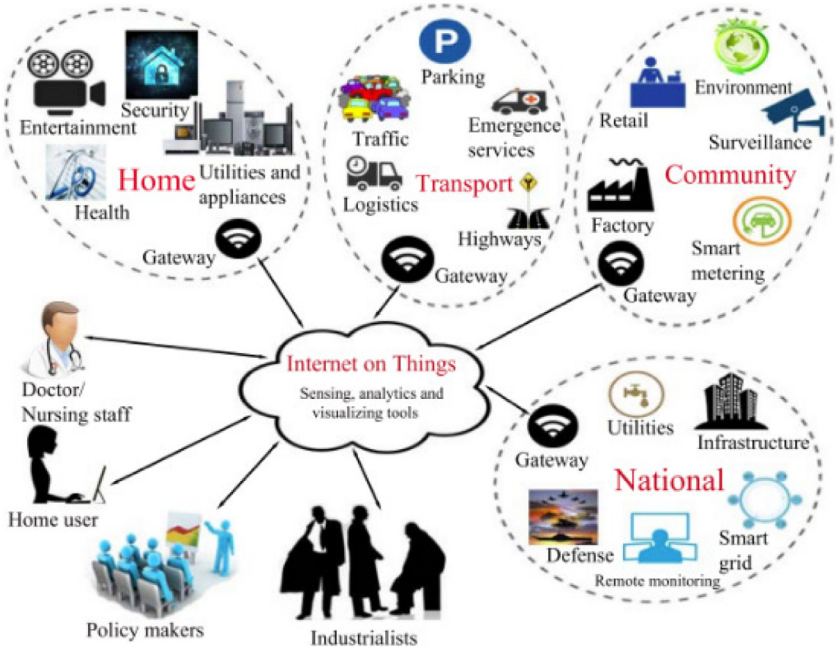



**Fig. 1 – A generic architecture of IoT Challa et al. (2017).**

tial attacks that an adversary could launch against our scheme and the related countermeasures. Section 6 evaluates the performance of our scheme and our conclusions are offered in Section 7.

## 1. Related work

Seshadri et al. (2004) proposed SoftWare-based ATTestation technique (SWATT), a code attestation protocol which is executed solely through software. Seshadri et al. also proposed another technique called SCUBA (Seshadri et al., 2006) to recover the sensor nodes after a compromise. This protocol uses two authenticated channels between a node and a base-station. Both protocols depend on accurate time measurements and optimal program codes to detect malware.

Recently, many hardware attestation techniques (Krauss et al., 2007; Tan et al., 2011) have been proposed, relying on the presence of a Trusted Platform Module (TPM) (Group, last access on 29/01/2016). A TPM can protect the computer system from malicious system activities and unauthorized changes. The tamper-proof memory of the TPM cannot be accessed by any entity other than the TPM itself. Each TPM has an Endorsement Key (EK) that uniquely identifies it. It is typically a public-private key pair; the private key is embedded in the TPM and never leaves it. Each TPM has the unique feature of secure Platform Configuration Registers (PCRs). These store metrics to measure the integrity of any code prior to its execution. This code can be the bootloader or application code. Whenever there are any alterations to this code, PCR values are extended with the hash digest of the changes, which can result in operation failures on some TPM functions. These functions are TPM_Seal and TPM_Unseal, which will be adopted in this paper to check the code image integrity. Tan et al. demonstrated that it is feasible to equip each sensor node with TPM and that the attestations can be coducted progressively from a base-station (Tan et al., 2011). However, (Tan et al., 2011) assumed that all devices are homogeneous and equipped with TPMs. This assumption does not hold for heterogeneous IoTs networks. Agrawal et al. recently proposed to verify program flash integrity in wireless sensor networks with cluster-heads equipped with TPMs (Agrawal et al., 2015). However, (Agrawal et al., 2015) only considered the single-hop setting and did not take into account of network attacks, e.g.,wormhole (Hu et al., 2006).

Tigist et al. proposed a Control-FLow ATtestation for Embedded Systems Software (C-FLAT) to defend against the runtime buffer overflow attack (Abera et al., 2016). Their attestation scheme is effective but C-FLAT cannot accommodate the control flow where there is an infinite loop as it will iteratively change the attestation response as the number of the loops increases. In MTRA, we compute the entire firmware as a hashed checksum. Therefore, even with an infinite loop in the control flow, it will not yield the same issue in MTRA.

Wazid et al. formulated authentication using three-way authentications for secure communication among the IoT nodes (Wazid et al., 2018). They adopted three components for authentication: user smart card, password and user biometrics. Except password, the overhead of smart card and user biometrics is too heavy for the embedded devices such as sensors. As to password, it is not secure to apply the same password to all the IoT devices. In addition, it is not scalable to manage the password if they are different from device to device.

Alqassem et al. further analysed the security and privacy requirements for IoT and surveyed a number of IoT applications in the context of security (Alqassem and Svetinovic, 2014). Most of these applications assume that the networks contain two types of devices: devices to interact with human

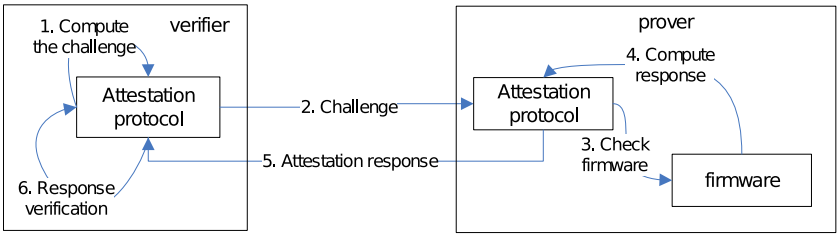



**Fig. 2 – General Remote Attestation process.**

and environment or devices serving as a gateway for data aggregation. We argued that the possible scenarios are much more complicated and the different security measures should be imposed in it.

Carpent et al. proposed a remote attestation via self-measurement in an unattended environment (ERAMUS) (Carpent et al., 2018b) and a further enhanced version (Carpent et al., 2018a). They adopted Sel4 (Klein et al., 2009), which is a bug-free operating system kernel, to assist the remote attestation when the self-measurement was processeed in Sel4. ERAMUS is efficient for the resource-constraint devices. However, it is an OS-dependent approach. If Sel4 is not compatiable with the IoT device, ERASMUS cannot be adopted for such IoT devices.

Hong et al. put forth an attestation protocol with Intel SGX for remote terminal and IoT (Wang et al., 2018). Such attestation protocol is tailored for a particular microcontroller (i.e., Intel SGX) and it is not a generic model that can be extended to other IoT devices.

Feng et al. developed a light-weighted remote attestation protocol called AAOT for low-resource IoT networks and CPS (Feng et al., 2018). It focuses on the power conservation but lacks the design to fully exploit the resources on the IoT devices with more computational resources.

## 2. System, threat model & our contributions

We assume that an IoT network contains three types of device. The first is a trusted third party that issues attestation challenges to the rest of the network (i.e., base-station). The device, directly controlled by the network administrator, has unlimited computational power and cannot be compromised by an adversary. There is only one such device in a network. The second type of device could accommodate tamper-proof hardware (e.g., TPM) but is vulnerable to some forms of network attacks. The third type has limited resources and lacks tamper-proof hardware. Thus, it is vulnerable to all forms of network attack. Moreover, we assume that the type-3 devices are one-hop neighbors of either the base-station or a TPM-enabled devices. This can be guaranteed with the cluster-head selection algorithm in wireless networks (Abbasi and Younis, 2007). In MTRA, the TPM-enabled devices are assumed to be the cluster-head to verify its one-hop neighbors without TPM. Therefore, MTRA is for the networks with fixed topology.

The general attestation process is shown in Fig. 2. The *verifier* computes the challenge according to the specified attestation protocol design, and sends it to the prover. The prover checks the firmware, computes the response and sends it back to the verifier. The verifier checks this response. If the response matches the expected result, the attestation succeeds.

We assume that an adversary can manipulate the software running on the second or third type of device but not on the first type. However, we also assume that an adversary cannot compromise devices physically, which means that the Read-Only Memory (ROM) and bootloader cannot be tampered with, as physical node compromise is not scalable for an adversary given a large number of devices in the same network. We also do not consider other types of Denial of Service (DoS) attacks except verifier-based DoS attacks (Brasser et al., 2016) (further discussed in Section 4.5), as the main goal of an adversary in remote attestation is to circumvent the program integrity check with malware running. Other DoS attacks such as jamming will result in the failure of attestation and reveal the existence of an adversary. However, adversary can intercept, replay, delay, forge or rush either the challenges as verifier or attestation response as prover against the second type of third type of device.

Our contributions are:

- *A new two-tier attestation protocol for heterogeneous IoT networks:* In prior work (Agrawal et al., 2015; Asokan et al., 2015; Seshadri et al., 2006; Seshadri et al., 2004; Tan et al., 2011), protocols were not designed in the context of heterogeneous IoT networks.
- *Countermeasures against rainbow and interference attacks:* In the challenge-response attestation protocol, the adversary attempts to send an arbitrary challenge to probe the response from the prover, which can be exploited for the Time-Of-Use to Time-Of-Check (TOUTTOC) attack without manipulating the prover's firmware. This is *rainbow attack*. In addition, an adversary can interfere with the challenge-response attestation protocol as a Man-In-The-Middle (Song et al., 2011). To the best of our knowledge, we are the first to adopt a one-way hash chain to defend against rainbow attacks. Each hash key is updated after the challenge is successfully completed. Therefore, the hash key used for one challenge is different from the key for the next challenge. This is further analyzed in Section 4.
- *Defense against wormhole attacks:* The wormhole attack is a notorious network attack that can allow an adversary to retrieve information earlier than expected for other types of attack (Hu et al., 2006). In our approach, a local one-way hash chain is proposed to invalidate the efforts of worm-

hole attacks. Section 4.3 details how our approach can defend against it. To the best of our knowledge, we are the first to defend against wormhole attacks in remote attestation.

- *New approach to detect Time-Of-Check-To-Time-Of-Use (TOCTTOU) attacks:* An adversary might have the correct firmware to respond to attestation challenges and have malware running in another period in a TOCTTOU attack. We deploy an online-offline notification mechanism to defend against TOCTTOU attacks in Section 4.4.
- *Randomization of flash regions to be verified:* In order to circumvent the remote attestation, adversary attempts to generate the correct attestation responses in advance for verification. We introduced a nonce in a challenge from verifier so that the response will not be the same. However, Song et al. introduced the MiM attack so that adversary can still construct the correct attestation response with nonces (Song et al., 2011). In MTRA, not only do we randomize the attestation response with the nonce, we also randomize the flash regions according to the value of nonces. In our earlier conference version (Tan et al., 2017), the randomization on memory region to be attested is not considered and evaluated. In this paper, we include this memory randomization technique in MTRA, which increased the entropy of the attestation responses. To the best of our knowledge, we are the first to propose the randomization on the flash regions to be verified in remote attestation design. In this way, adversary cannot forge or collect the possible correct attestation responses off-line to assist the TOCTTOU attack.

## 3. Remote attestation design

There are two stages for the proposed attestation protocol:

- *Offline (preparation stage)*, when none of the devices have been deployed for operations and related additional hardware (i.e., TPMs) is initialized and installed on the computationally powerful devices. As no network has not been established, no adversary can launch an attack.
- *Online (operative stage)*, when all devices have been deployed and are operational. The remote attestations are performed and an adversary can launch attacks.

### 3.1. Notations

The notation in the rest of this paper are shown in Table 1.

**Table 1 – Notations.**

| Notation | Meaning |
|---|---|
| $h(M)$ | The one-way hashed result on message M |
| $A \rightarrow B: M \leftarrow P$ | A sends a packet P to B<br>packet P's content is copied to M |
| $E_{Key}(M)$ | Encryption of message<br>M with *Key* |
| $D_{Key}(M)$ | Decryption of M with *Key* |
| $\|P\|$ | The length of message P |
| $P\|\|M$ | Message P is concatenated with message M |
| $M_1\%M_2$ | The remainder of message $M_1$ divided by $M_2$.<br>The result will vary from 0 to $M_2 - 1$. |

### 3.2. Offline (Preparation stage)

In this stage, we assume that it is secure to distribute initial shared secrets among all nodes prior to deployment. For example, we assume that a trusted key distribution server is available to distribute the cryptographic information securely. The initial code image is installed on all devices.

The TPMs on the TPM-enabled devices are initialized with the asymmetric key pair, $L_{pub}$ and $L_{pri}$, which is bound to PCRs and stored in the TPM. The configurations of the devices are extended into the PCR. The related TPM operations will fail if the configurations of the devices (e.g., the contents in program flash) are altered. The hash of firmware is sealed with the $L_{pri}$ in TPM. A trusted server can distribute $L_{pub}$.

A one-way hash chain is created (see Fig. 3) Hu et al. (2005). The last hash key (i.e., the committed value of hash chain, $V_k$[1]) is distributed to all devices. $V_{k-i}$ is to authenticate the $(i+1)^{th}$ challenge from the verifier. Only the base-station keeps all the elements in this hash chain.

The number of key chain elements should be sufficient to cover the lifetime of the IoT network. However, if the number of key chain elements is not sufficient, a new key chain is generated in the base-station and its committed value can be distributed to all the devices. he committed element of the one-way key chain (i.e., $V_0$ in Fig. 3) can be made known to public as only the base-station has previous key (e.g., $V_1$ in Fig. 3) for key verification.

After the application code image is installed, the free space in the program flash of devices without TPMs is filled with unique, random incompressible bits (Fig. 4). Each device is filled with a unique bit pattern (Agrawal et al., 2015).

### 3.3. Online (Operative stage)

In this stage, all the devices are deployed and become operative. When the base-station needs to verify code images for the networks, it first contacts one of the one-hop neighbours with TPM equipped as an initiator for attestation. The challenge is $E_{V_{k-i+1}}(N_i||V_{k-i})$ (line in Algorithm 1) where $N_i$ is a fixed-length random nonce generated by the verifier and $i$ denotes that it is the $i^{th}$ attestation since deployment and $k$ is number of keys in one-way hash chain in Fig. 3.

On receipt of the challenge, the prover will decrypt the packet with an unsealed hash key (line in Algorithm 1). If it matches the one-way hash property, the prover will further unseal the hashed result of the code image. If the program flash had been altered, the unseal operation in line in Algorithm 1 will fail, since the PCRs in TPM will have been extended with the changes in program flash. In MTRA, only partial program flash is checked. The range of the program flash to be verified is between Nonce%(flash size) and $h$(Nonce)%(flash size), which are the remainders of nonce and

[1] Shown as $V_k$ in Fig. 3, where $k$ denotes the upper limit of the number of attestations in the entire lifetime of the network.

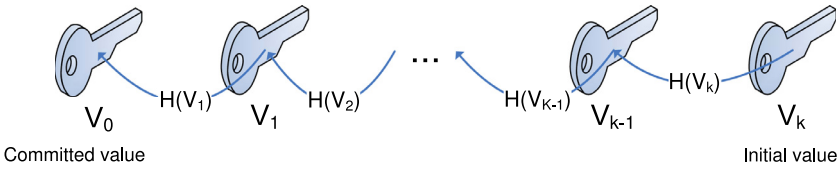



**Fig. 3 – One-way hash chain.**

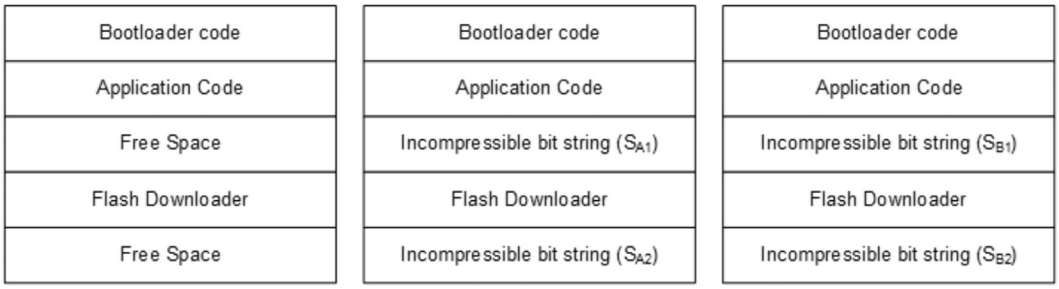



**Fig. 4 – Program flash distribution in devices without TPMs: the rest of the free space is filled with unique random incompressible bits. A and B are filled with different bit pattern in the free space i.e., $S_{A1} \neq S_{B1}$ and $S_{A2} \neq S_{B2}$.**

**Algorithm 1** The ith Remote Attestation for TPM-enabled devices.

**Require:** A knows $V_{k-i}$ and B knows $V_{k-i+1}$
1: A → B: $M \leftarrow E_{V_{k-i+1}}(N_i||V_{k-i})$
2: $B : TPM\_Unseal(V_{k-i+1}, L_{pri})$
3: B:$N'_i$, $V'_{k-i} \leftarrow D_{V_{k-i+1}}(M)$
4: **if** $H(V'_{k-i}) \neq V_{k-i+1}$ **then**
5: B → A: $M \leftarrow E_{V_{k-i+1}}(N'_i)$||'key chain error'
6: **else**
7: $B : TPM\_UnSeal(h(\text{firmware}_B), L_{pri})$
8: **if** *TPM_UnSeal* is successful **then**
9: B: remove $V_{k-i+1}$, keep $V_{k-i}$
10: B: Firmware ← Flash(Nonce%(flash size), $h$(Nonce)%(flash size))
11: $B \rightarrow A : M \leftarrow E_{V_{k-i}}(N'_i||h(\text{Firmware}))$
12: $B : TPM\_SEAL(V_{k-i}, L_{puc})$
13: A:$N'_i$, $h(\text{firmware}_B)' \leftarrow D_{V_{k-i}}(M)$
14: **if** $N'_i = N_i$ and $h(\text{firmware}_B)'$matches with expected result **then**
15: attestation of B succeeds
16: **else**
17: **if** "key chain error" nessage is received **then**
18: restart the attestation.
19: **else**
20: attestation of node B fails
21: **end if**
22: **end if**
23: **else**
24: B → A: $M \leftarrow E_{V_{k-i+1}}(N_i||$"attestation fails")
25: **end if**
26: **end if**
27: **if** A does not receive any response from B for an extended period of time **then**
28: attestation of B fails
29: **end if**

its hashed value, divided by the memory size. This mechanism confines the memory section boundary between 0 and flash size − 1 so that the range of flash regions being checked is randomized by challenge nonce and it will be different in each attestation (line and in Algorithms 1 and 2). In this way,

**Algorithm 2** The *ith* Remote Attestation for non-TPM devices.

**Require:** A knows $V_{k-i}$ and B knows $V_{k-i+1}$
1: A → B: $M \leftarrow E_{V_{k-i+1}}(N_i||V_{k-i})$
2: B:$N'_i$, $V'_{k-i} \leftarrow D_{V_{k-i+1}}(M)$
3: **if** $H(V'_{k-i}) \neq V_{k-i+1}$ **then**
4: B → A: $M \leftarrow E_{V_{k-i+1}}(N'_i)||$"key chain error"
5: **else**
6: B: Firmware ← Flash(Nonce%(flash size), $h$(Nonce)%(flash size))
7: A $N'_i$, $h$(firmware) $\leftarrow D_{V_{k-i}}(M)$
8: **if** $N'_i = N_i$ and $h(\text{firmware})'$matches the expected result **then**
9: B: remove $V_{k-i+1}$, keep $V_{k-i}$
10: attestation of B succeeds
11: **else**
12: **if** "key chain error" nessage is received **then**
13: A: restart the attestation.
14: **else**
15: A: attestation of B fails
16: **end if**
17: **end if**
18: **end if**
19: **if** A does not receive any response from B for an extended period of time **then**
20: attestation of B fails
21: **end if**

adversary is not able to compute a correct firmware response in advance as the flash regions to be verified are not known before challenges are sent out. In the rest of this section pseudocodes, node A is the verifier and node B is the prover unless specified.

The new hashed key is updated and sealed for the next attestation (lines - in Algorithm 1). The prover will send the encrypted nonce and hashed digest of the entire code image back to the verifier for attestation (line in Algorithm 1). If the

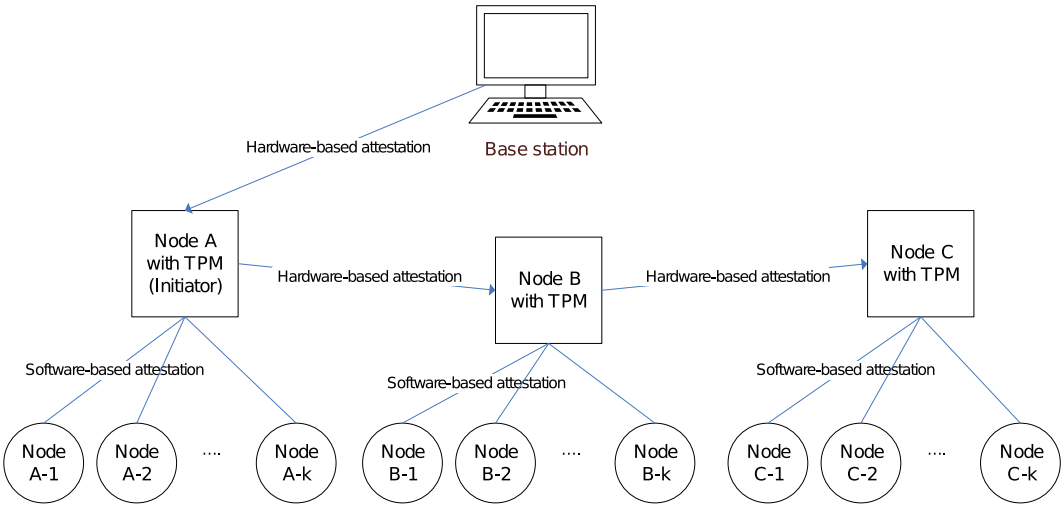



**Fig. 5 – The remote attestation structure in heterogeneous embedded devices: Nodes A, B and C are equipped with TPMs while node$_{A/B/C_n}$ are without TPM.**

decrypted nonce is the same as the one in the challenge and the hash digest of the code image is as expected, the attestation succeeds (line in Algorithm 1).

In addition, if the verifier does not receive any response from the prover for an extended period, the verifier will assume that the prover might have been compromised or might not be operational. Therefore, the attestation will fail. The initiator can further verify other TPM-enabled nodes, as was proposed in Asokan et al. (2015). The above process can be iteratively expanded to all other TPM-enabled devices (see Fig. 5), until all TPM-enabled devices have been attested.

For provers without TPM, the verifier could send them the same challenge[2]. The prover will follow a process similar to that described in Algorithm 1. However, without TPM functions, The flash regions to be verified are randomized based on the nonce (line in Algorithm 2). Therefore, the attacker cannot hide malware in the program flash of the compromised device as the memory where the malware is located could be checked by the verifier. An adversary cannot manipulate the program counter to point to the healthy code image during attestation (Time-Of-Check case) and return to malware (Time-Of-Use case) at other periods. The same nonce , concatenated with the hash digest of the code image, is sent back to the verifier, encrypted with the updated hash key. Instead of computing the hash digest of the code image, the verifier will check the hash digest of the entire program flash for these devices without TPM. Algorithm 2 illustrates this attestation process.

Each device without TPM is attested by its TPM-enabled neighbors until all devices have been verified.

When a device is connected or disconnected from the network, it must perform the operations in Algorithm 3. Each device must report to its previous verifier if it is put online or taken offline. This procedure is deployed to defend against TOUTTOC attack with additional devices, which is further discussed in Section 4.4.

[2] The challenge is revised to defend against wormhole attack, which is described in Section 4.3.

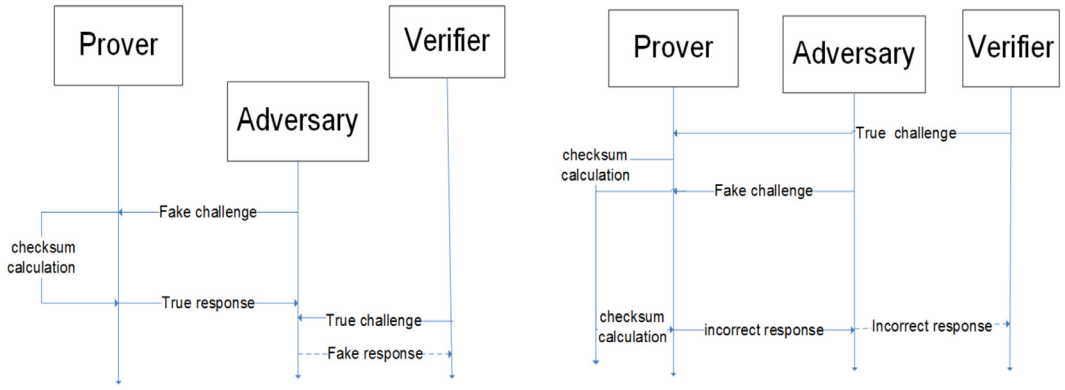
**Algorithm 3** prover device joins or leaves the networks after the *ith* successful attestations for *k* times.

**Require:** A knows $V_{k-i}$ and B knows $V_{k-i}$
B is to be offline: B → A: $M \leftarrow E_{V_{k-i}}(Nonce_k||\text{offline message})$
**if** A is not waiting for an attestation response from B **then**
  A : Nonce $'_k$, *offline message* $\leftarrow D_{V_{k-i}}(M)$
  **if** $Nonce'_k = Nonce_k$ **then**
    A: Store $Nonce'_k$
  **else**
    A: B is not authenticated
  **end if**
**else**
  A: B is compromised or abnormal
**end if**
...
B is online: B → A: $M \leftarrow E_{V_{k-i}}(Nonce_k||\text{Online message})$
**if** A is not waiting for an attestation response from B **then**
  A : Nonce $'_k$, *online message* $\leftarrow D_{V_{k-i}}(M)$
  **if** $Nonce'_k = Nonce_k$ **then**
    A: Store $Nonce'_k$
  **else**
    A: B is not authenticated
  **end if**
**else**
  A: B is compromised or abnormal
**end if**

## 4. Security analysis

### 4.1. *Man-In-the-Middle Attack (MIM attack)*

As shown in Fig. 6, an attacker can intervene between the verifier and prover, which is known as a Man-In-the-Middle attack. It can lead to two potential threats during the attestation process: *rainbow attacks* and *interference attacks*. In a rainbow attack, an adversary keeps sending arbitrary challenges to the prover for correct responses so that it can construct a challenge-response pair table (Fig. 6a). An interference attack disturbs the challenge-response process by leading the prover to send an incorrect response to the verifier (Fig. 6b).




(a) Rainbow Attack. (b) Interference Attack.

**Fig. 6 – Man-in-the-Middle attacks.**

Song et al. proposed a One-way Memory Attestation Protocol (OMAP) to eliminate the challenges from the verifier (Song et al., 2011). An adversary cannot forge the challenges. However, OMAP is subject to a false attestation response attack. An adversary can readily intercept the correct attestation checksum from a prover and replay it to circumvent the attestations for other compromised network devices without the nonces. In our approach, we adopt the one-way hash key to encrypt the challenge (line in Algorithm 1). Even though an adversary might learn about the hash key to encrypt the challenge (i.e., $V_{k-i+1}$ in Algorithm 1) by compromising the devices in the IoT network, it still cannot construct a valid challenge to request a response from a prover, as $V_{k-i}$ is not known to any nodes to construct a valid challenge ($E_{V_{k-i+1}}(N_i||V_{k-i})$ in Algorithm 1) until the next challenge is sent out. The prover will ignore the challenges if the one-way hash relationship cannot hold between $E_{V_{k-i+1}}$ and $E_{V_{k-i}}$ (line in Algorithm 1). It can also defend against a replay attack, in which an adversary stores a valid attestation challenge or response and replays it later at a time of its own choosing (Kil et al., 2009), as the one-way key will have been updated later (line in Algorithm 1 and line in Algorithm 2). Therefore, replaying an outdated attestation challenge or response cannot circumvent the one-way key chain check. An adversary might launch a wormhole attack to replay the challenge remotely, which is discussed in Section 4.3.

### 4.2. *Impersonation & replay attack*

In MTRA, a challenge is encrypted with a session key ($V_{k-i+1}$ in line of Algorithm 2 or line of Algorithm 1). The key is replaced with its next element in the one-way hash chain so a captured challenge cannot be replayed as the next attestation request because the key to decrypt the challenge is different. Therefore, replay attack is not feasible in MTRA. In addition, the memory to be attested is different in each attestation. So the attestation response is different. The impersonation attack is similar to Man-in-the-Middle attack as the verifier or prover can be a d masqueraded user but they cannot forge a valid session key for the challenge (as a verfier) or the response (as a prover). Therefore, neither attack cannot be launched against MTRA unless the session key and the ransomized memory regions to be attested are known in advance.

### 4.3. *Wormhole attack*

To circumvent the hash value check in challenges in Section 4.1, the adversary captures the challenges from the base-station, obtains $V_{k-i}$ by challenge decryption using the $V_{k-i+1}$ through node compromise, tunnels it to another area of the network, where $V_{k-i}$ is not disclosed, and reconstructs a valid challenge. It is called wormhole attack (Hu et al., 2006) (Fig. 7).

In the IoT, it is easier to defend against wormhole attacks between TPM-enabled devices only. An adversary cannot retrieve the hash key through compromising a TPM-enabled device because of the tamper-proof hardware[3]. It can only intercept the challenge packets and brute-force the hash key online as the hash key is updated with the next element in the one-way hash chain upon the verification of challenges. However, an adversary could compromise non-TPM devices to launch a wormhole attack. We propose a local one-way hash chain to defend against wormhole attacks as follows:

Each TPM-enabled device maintains another distinct local one-way hash chain, denoted as $L$. This hash chain is only used between the given TPM-enabled device and its one-hop neighbors. The committed value of this hash chain is $L_k$, where $k$ is the length of the chain. $L_k$ is distributed when the non-TPM devices establish the one-hop neighbourship with TPM-enabled devices. The key for encrypting the $i$th challenge ($P_i$) becomes $V_{k-i+1} \oplus L_{k-i+1}$. The encrypted challenge is $E_{P_i}(N_i||V_{k-i}||L_{k-i})$. The non-TPM device can decrypt this package and retrieve $L_{k-i}$ to check the one-way hash relationship between $L_{k-i}$ and $L_{k-i+1}$ before generating the response. An adversary can retrieve $L_{k-i}$ but it is not a valid local hash key if it is used in another area of the network. A different TPM-enabled device will hold a different local hash chain so the

[3] An adversary can still capture a TPM-enabled device but the system-related TPM commands will fail and the attestation result will expose the capture of the TPM-enabled device.

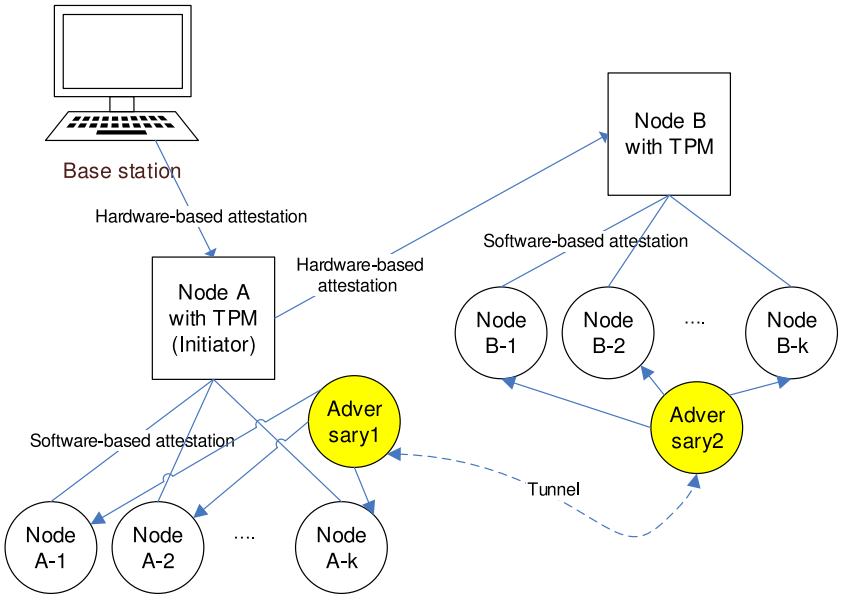



**Fig. 7 – Wormhole attack: two adversaries establish a low-latency tunnel to exchange the packets they overhead for replay.**

tunnel will not facilitate rainbow or interference attacks, remotely. The response will also be encrypted with $V_{k-i} \oplus L_{k-i}$ so that the replayed attestation response through the tunnel in the other area will not be accepted, as $L_{k-i}$ would be different if the packets were tunnelled to a different network area.

### 4.4. *Time-Of-Check-To-Time-Of-Use (TOCTTOU) Attack*

In a TOCTTOU attack, an adversary will use the legitimate firmware to compute the correct checksum for the response on the receipt of the challenge (TOC case) and run the malware at other time (TOU case). When a TPM-enabled device is compromised, the key to unseal the checksum of the code image will fail as the system configuration has been changed (line in Algorithm 1). If the TPM-enabled device is compromised, the sealed secrets cannot be retrieved to construct a correct response. For non-TPM devices, the program flash, except for the code image section, is filled with random pre-set bytes. No extra room is available for malware.

An adversary could deploy an additional device to store the correct firmware for attestation, given that no memory space is available for malware on the compromised device. On receipt of the challenge, the adversary puts the compromised devices with malware offline and sets the additional device running with correct firmware online (Time-Of-Check case). There is an ID conflict if two devices with the same ID (one with malware and another with correct firmware) are presented in the network at the same time. The device will send out the message confirming it is online (line in Algorithm 3). The verifier will know the prover is back online before receiving the attestation response. This prover is marked as 'compromised' as the notification of being online is not expected during the challenge-response process. If an adversary modifies the codes so that the online notification is not sent, the checksum of the firmware could not pass the verification. Therefore, even though the adversary is able to deploy additional hardware to respond to the attestation challenge correctly, the substitution cannot evade detection by the verifier. It is the same procedure when the device with the correct firmware is offline (line in Algorithm 3), although an adversary could readily switch off the device before it sends out the offline message. However, switching off the compromised device will not help pass the attestation; if the verifier does not receive any response from a prover within a given period, the verifier will assume that the prover is faulty or compromised (line in Algorithm 1).

### 4.5. *Verifier-based DoS attack*

Ferdinand et al. recently proposed that an adversary could send an excessive number of attestation challenges to other legitimate resource-constrained devices, that are then fully occupied in fulfilling the attestation challenges, rather than performing the expected functions such as sensing or actuating (Brasser et al., 2016). Such a Denial of Service (DoS) attack is called a verifier-based DoS attack. This attack is based on the assumption that the overhead of preparing an attestation challenge is significantly smaller than that of preparing an attestation response. In our scheme, the malicious verifier must find the correct one-way hash key to make the prover run the entire attestation process. (i.e., $V_{k-i}$ in Algorithm 1 for the TPM-enabled device for the $i+1^{th}$ challenge). The prover will not process the attestation challenges if the verification of one-way hash key does not pass (i.e., 'key chain error' in Algorithm 1). The prover only requires one hash operation and symmetric decryption to check, so it cannot incur much overhead to cause the DoS effects. Even with an excessive number of attestation challenges, the impact of verifier-based DoS attacks is minimal as the symmetric decryption and hash operations are light-weight.

## 5. Implementation details

In the conference version (Tan et al., 2017), the TPM implementation details were not discussed in details due to limited pages available. In this manuscript, we would further detail the TPM commands adopted in MTRA. There are three types of TPM commands: the shared commands used by verifiers and provers, the TPM commands used by verifiers only and those used by provers only.

The following are the common commands executed by verifiers and provers during remote attestation.

- Tspi_Context_Create() generates the TrouSerS context to communicate with TPM
- Tspi_Context_Connect() connects TPM with TrouSerS
- Tspi_Policy_GetPolicyObject() returns the respective TPM policy of the root key to generate the Attestation Identity Key (AIK) to encrypt or decrypt the challenge or attestation response.
- Tspi_TPM_CreateObject() creates an object that can be stored in the hard drive for the verifier or prover. It refers to the data exchanged between the verifier and prover.
- Tspi_TPM_PCRRead() returns the value of the Platform Configuration Registers(PCRs). If the firmware is altered, the value in a PCR is updated with a new value. The attestation response is bound to the PCR, so any changes in the firmware will lead to a mismatch between the PCR values of the verifier and prover and the attestation response.
- Tspi_Hash_GetHashValue computes the hashed checksum of the firmware. It is called when the challenge is successfully decrypted and verified with the one-way hash function between $V_{k-i}$ and $V_{k-i+1}$.
- Tspi_Policy_LoadKey() returns the key object from the given policy.
- Tspi_TPM_Unseal() decrypts the messages associated with the specified PCR values. The verifier executes this command to check the attestation response while the prover retrieves the firmware hash internally.
- Tspi_Context_FreeMemory() frees the memory of the TrouSerS context. it is only executed when the TPM commands have been all completed.
- Tspi_Context_Close() closes the TrouSerS session.

These common commands have the same overheads for the verifier and prover if they are executed on Odroid Xu4. Verifiers and provers have their own sets of commands.

The following commands are executed by the verifier only.

- Tspi_Policy_SetSecret() creates the shared secret between the verifier and prover. In our scheme, it is the initial value to generate the one-way hash key chain (i.e., $V_0$ in Fig. 3). After it is hashed $T$ times, the committed value of the hash chain is transmitted to the prover.
- Challenge encryption for the nonce and the hashed key used for the next challenge (i.e., $E_{V_{k-i+1}}(N_i||V_{k-i})$ in Algorithm 1). The symmetric encryption is not included in TPM operations so they can be implemented in other cryptographic package.
- Tspi_Key_Createkey() creates the AIK. The private key is used to sign the challenge from the verifier while the public key is attached to the challenge so that the prover can encrypt the attestation response.
- Tspi_Key_RegisterKey() registers the public key from AIK with the specific data object. This data object is transmitted to the prover, which can retrieve the public key with Tspi_Key_GetPubKey().

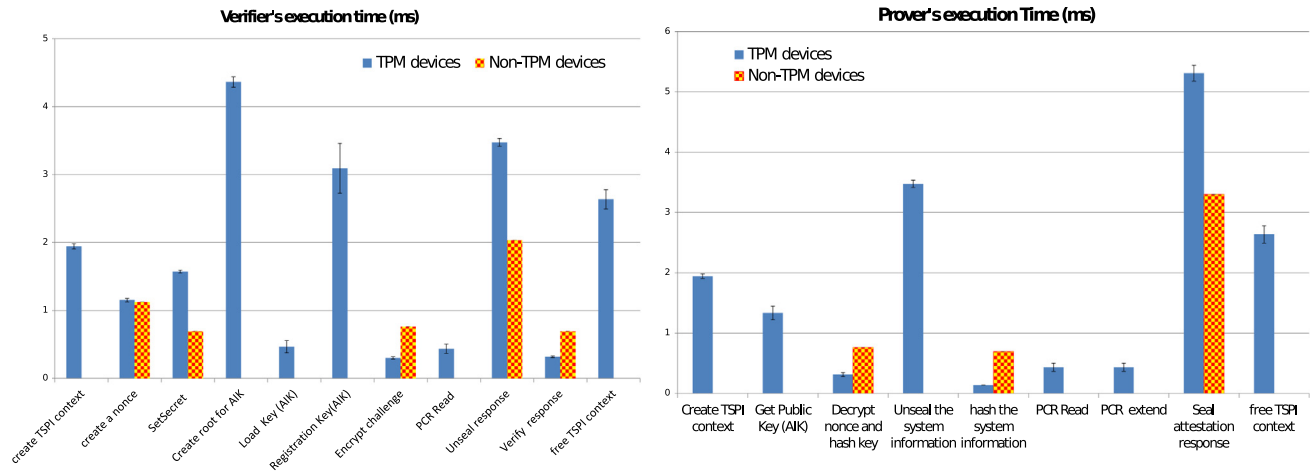
The following commands are executed by the prover only.

- Challenge decryption for the nonce and the hashed key used for the next challenge (i.e., $D_{V_{k-i+1}}(M)$ in Algorithm 1).
- Tspi_Key_GetPubKey() retrieves the public key from the verifier for the seal command
- Tspi_TPM_Seal() encrypts the attestation response, associated with the specific PCRs.
- Tspi_TPM_UnSeal() unseals the sealed commands used by the prover to generate the attestation response back to the verifier. The unseal command is used internally to retrieve the hashed firmware image associated with system state when it is sealed.
- Tspi_PCR_Extend() updates the PCR values accordingly so that the attestation response cannot be replayed by an adversary even if the one-way hash chain is compromised. This operation is not reversed.

## 6. Performance evaluation

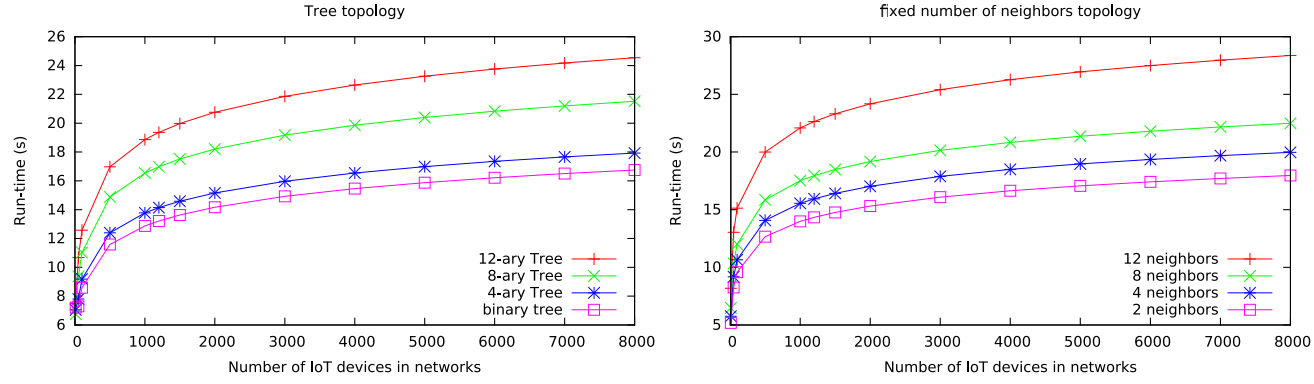
We implemented MTRA on two types of single-board computer. For TPM-enabled devices in Fig. 5, we employed an Odroid-XU4 single-board computer. It gives excellent computing power for its size. This versatile 8-core ARM-Linux computer runs a fully fledged Ubuntu, which can support a software-based TPM emulator (Strasser and Stamer, 2008) and TrouSerS (Trusted Computing Software Stack) (Tro, 2008, last accessed: 13/09/2016). For the devices without TPMs in Fig. 5, we used Raspberry PI Model B+ Anderson (2014), a 4-core ARM-linux single-board computer. We adopted BouncyCastle (BC) (bou, 2009, last accessed on 16/09/2016) as our cryptographic API for the non-TPM devices (i.e., Raspberry PIs). The cryptographic operations we use are the hash commands and symmetric encryption and decryption. We also use these operations for hash and symmetric cryptographic operations for TPM-enabled devices (i.e., Odroid-XU4) as they are used to encrypt the challenges for provers.

We measured the execution time of each TPM command and cryptographic operation on both types of device in Section 6.1. Then we simulated MTRA for large-scale networks using the Network Simulator 3 (NS3) (NS3, Feb. 2011, last accessed on 18/04/2017). We implemented MTRA at the application layer and employed the latencies based on the measurements in Section 6.1 for the simulations of the corresponding cryptographic operations to evaluate the scalability of MTRA in Section 6.3 . The communication data rate for the links between two devices (either TPM or non-TPM ones) was set to 250 Kbps, which is the defined data rate of Zigbee, a common protocol used among the IoT devices . Then we further




(a) Verifier. (b) Prover.

**Fig. 8 – Remote attestation command execution times in TPM-enabled devices.**




(a) Tree Topology. (b) Fixed number of Neighbors.

**Fig. 9 – Scalability Test of MTRA: the attestation process is more cost-efficient in tree topology than the fixed number of neighbors.**

simulated and evaluated the prover-based attack (i.e., TOT-TOU attack) in Section 6.4 and the verifier-based attacks (i.e., MIM attack, wormhole attack and Verifier-based DoS attack) in Section 6.5. In addition, we have computed the communication overhead and compared MTRA with other remote attestation protocols in Section 6.7.

### 6.1. *Hardware execution time*

Fig. 8 shows the execution time for each operation during remote attestation in our scheme. For commands available in both devices, the Odroid-XU4 (TPM-enabled devices) requires 50% more time than the Raspberry Pi (Non-TPM devices). The latency in TPM-enabled devices is attributed to the communications between the processes of the TPM emulator and TrouSerS, an interface between TPM and application. The TPM-enabled devices have more commands to execute due to the TPM environment setup (e.g., TrouSerS context creation). The exceptions among these commands are nonce creation and encryption, as both types of devices use the same set of BouncyCastle APIs and Odroid-XU4 is more powerful than raspberry Pi. TPM devices therefore perform better than non-TPM ones. The total execution time required is in the order of milliseconds. In MTRA, the verifier is required to perform more tasks than the prover. The verifier-based DoS attack (Brasser et al., 2016) is more difficult to launch since the assumption of the verifier's overhead being much lighter than that of the prover does not hold.

### 6.2. *Communication overhead*

During the $i$th remote attestation with a $k$-element key chain, the followings messages are exchanged:

- In the initialization phase, the verifier and prover will establish the shared secret for the one-way hash key chain (i.e., $V_k$) as the committed value of the key chain. The length of this hash value is 20 bytes (SHA , 2001, last accessed on 16/09/2016).
- $E_{V_{K-i+1}}(N_i||V_{k-i})$ in Algorithm 1, which is a challenge sent to prover. The nonce and the one-way hash key each require 20 bytes (SHA , 2001, last accessed on 16/09/2016). There-

fore, the communication overhead is 40 bytes. It is the same with or without TPM.

- If a key chain does not match, the prover will send back a key chain error message, as well as the nonce ($E_{V_{k-i+1}}(N_i'||'\text{keychainerror}')$). The size of the packet is 21 bytes (i.e., the prover and verifier agree to use a one-byte operation code to denote these errors, in addition to the 20-byte nonce).
- After the prover computes the checksum of the firmware, it will send back the attestation response. The size of the response is 40 bytes (i.e., $E_{V_{k-i}}(N_i'||h(\text{firmware}))$ in Algorithm 1). There is one 20-byte nonce in addition to a 20-byte firmware hash.

Therefore, for one successful attestation process, the communication overhead is 40 bytes in total. For devices leaving or joining the networks, the communication overhead is 21 bytes. Consequently, MTRA has minimal communication overhead compared to algorithms that use the ECDSA signature (the signature alone is more than 256 bytes) scheme to verify the program hash (Asokan et al., 2015). We use the one-way property to ensure an adversary cannot retrieve the next hash key before it is used. This makes our attestation protocol more suitable for power-constrained embedded devices. Even with a separate hash chain to defend against wormhole attacks, the additional communication overhead is another 20 bytes for the hash value, which makes the overhead 60 bytes.

### 6.3. *Scalablity test*

We simulated MTRA with different network topologies, including trees, assuming that the root of the tree is the base-station and the fan-out degree is 2,4,8 and 12 or the networks with a fixed number of one-hop neighbors (4,8 and 12). In both topologies, the ratio between non-TPM devices and TPM enabled devices was the same as the fan-out degree or number of one-hop neighbors (e.g., in a 500-device network with fan-out degree of 4, the number of TPM enabled devices was 100, while the number of non-TPM devices was 400). We also varied the network size from 10 to 8000 devices. We measured the runtime from when the base-station initialized the remote attestation process until the remote attestation process completed on all devices. The simulation results showed that the attestation latency was logarithmic in relation to the size of the network. In particular, the remote attestation in the tree topology was completed with 10 – 15% less latency than it was in the topology with a fixed number of neighbors. The reason was that the attestation process of MTRA was similar to breadth-first search. Thus, the attestation process in tree topology was more cost-efficient than the topology with a fixed number of neighbors, where the challenges were sent to the same device from different neighbors.

### 6.4. *TOCTTOU attack evaluation*

TOTTOU attacks assume a malicious prover. We were interested in how long the base-station can identify the compromised prover(s). Therefore, we focused on how the percentage of compromised provers would affect the time taken to identify these compromised provers. In our simulation, the network size was fixed at 1,000 devices and we varied the ratio of TPM devices and non-TPM devices. We measured the latencies from the time when the base-station initialized the remote attestation until the time when all the compromised provers were identified by either TPM-enabled verifiers or the base-station. If the compromised prover were a TPM-enabled device, its attestation response could not be retrieved from TPM correctly due to the configuration change of the compromised system. A TOCTTOU attack could then be identified immediately. For the non-TPM devices, the adversary puts the malware offline for check and online for use, which would trigger the on-off messages sent to the verifier, who can identify the compromised provers. This would incur some communication overhead. We varied the percentage of compromised provers from 10% to 90% and those compromised devices were randomly distributed during the simulation. Figure 10 shows the time taken to identify the compromised provers in terms of percentage of TPM-enabled devices and compromised devices in the networks. When the percentage of TPM-enabled devices is more than 40%, MTRA can identify the compromised prover with a negligible latency. Given the same percentage of TPM-enabled devices, identifying 10% compromised provers took the most time as those compromised provers were sparsely distributed in the networks, delaying on-off messages sent back to verifier (line and in Algorithm 3). When the percentage of compromised devices reaches 90% in the networks, more on-off message transmissions were triggered in networks,incurring significant identification latency.

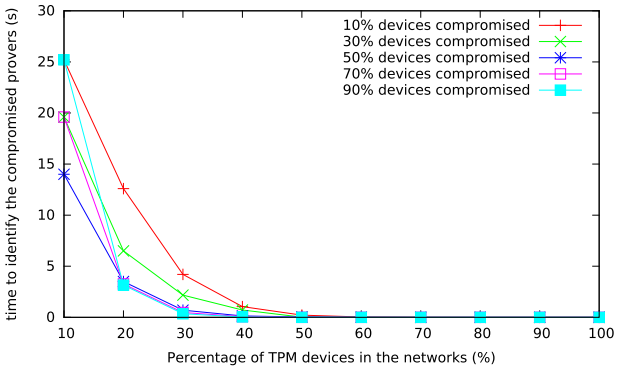



**Fig. 10 – Time to identify the TOTTOU attack in MTRA.**

### 6.5. *Verifier-based attack evaluation*

MIM attacks, wormhole attacks and verifier-based DoS attacks assume a malicious verifier. To defend against these verifier-based attacks, the provers will authenticate the challenges. In MTRA, we adopted the one-way hash chain to authenticate attestation challenges. For the verifier, it encrypts the challenge nonce and the next hash key with the current hash key (line in Algorithms 1 and 2). For the prover, it will firstly decrypt the challenge, then check whether the one-way hash relationship holds between the current hash key and the next hash key (line in Algorithm 1 and line in Algorithm ). As a result, the prover has two cryptographic operations to authenticate the challenges: one is the symmetric decryption. The other is the one-way hash operation. The wormhole attack is a form of

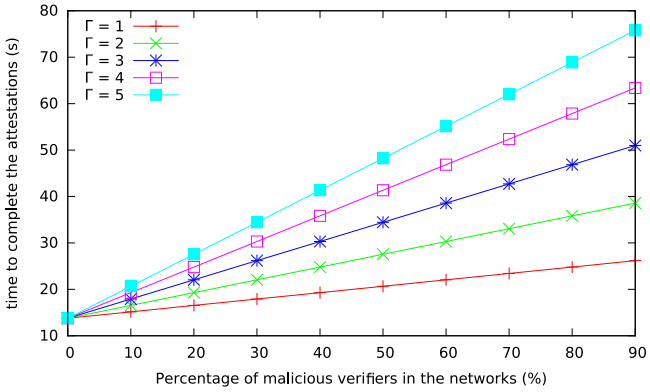



**Fig. 11 – Attestation time in presence of MIM attacks, wormhole attack or Verifier-based DoS attack: $\gamma$ indicates how many invalid challenges the provers have received before they will stop responding the verifier's further challenges.**

MIM attack, tunneling the challenges to another network area. MTRA defends against it by deploying the local one-way hash chain. The corresponding overhead is the same as with the MIM attack. It is also the same for the Verifier-based DoS attack: the malicious verifier keeps sending out the invalid challenge to deplete the resources of provers. We simulated these verifier-based attacks by using a set of malicious verifiers to keep sending the arbitrary attestation challenges. Then we set a threshold (denoted as $\gamma$, varied from 1 to 5) for the number of invalid challenges before provers completely ignore the challenges from the same verifier. We varied the percentage of malicious verifiers from 0% to 90% within a fixed 1,000-node networks with a tree topology whose fan-out degree is 8. We measured the time taken to complete one attestation process from base-station until all other legitimate devices were attested except those malicious verifiers. Fig. 11 illustrates the attestation time in simulation. The time taken to complete attestation increases in linear scale to the percentage of the malicious verifiers in the networks. The execution time taken to verify one challenge is fixed in MTRA. The attestation time only increases when the number of challenges from different verifiers increases if the number of invalid challenges before provers stop responding the respective verifiers' further challenges ($\gamma$ in Fig. 11) is fixed. The $\gamma$ in these three attacks is a key design parameter in MTRA. If we wish to achieve the best performance, $\gamma$ should be 1 but it can introduce some false positives for verifiers, which might cause the prover to be identified as'compromised' if it stops responding that verifier. There is a trade-off between defending mechanisms against the attacks from provers and those from verifiers. If the provers are more subject to attacks, $\gamma$ can be set to a higher value to minimize the false positives and vice versa.

### 6.6. *Memory range randomization evaluation*

In this section, we further evaluated the impact of the memory region randomizations on MTRA. In the conference version, the memory region to be attested is always the same memory region (Tan et al., 2017). In this manuscript, the memory region is randomized and the size of the memory region to be attested varies in each attestation. The memory regions to be verified are randomly chosen based on the challenge nonce. The smaller the size of expected randomized memory region to be verified is, the more difficult it is for adversary to guess the correct memory regions to be verified for TOCTTOU attack. For the expected memory region to be verified, we have the following theorem:

**Theorem 6.1.** *If the memory regions to be verified are randomly and independently chosen, the expected average memory size to be verified is 1/3 of the total memory size.*

**Proof.** Let $X$ be a random variable uniformly distributed over $[0, L]$, where $L$ denotes the total size of the memory region and 0 indicates the starting address of the entire memory region. The Probability Density Function (PDF) of $F(X)$ is given by

$$F(X) = \begin{cases} \frac{1}{L} & X \in [0, L] \\ 0 & \text{Otherwise.} \end{cases} \tag{1}$$

Let us randomly pick two memory addresses within [0,L] as the upper and lower limit of the memory region to be verified, denoted as $X_1$ and $X_2$. They are random variables distributed according to $F(X)$ in Eq. (1). The size of the memory region between these two memory addresses are a new random variable

$$Y = |X_1 - X_2| \tag{2}$$

Hence, we would like to find the expected value $\mathbb{E}(Y) = \mathbb{E}(|X_1 - X_2|)$ in relation to $L$. Let us introduce function g as follows:

$$g(X_1, X_2) = |X_1 - X_2| = \begin{cases} X_1 - X_2 & X_1 \geq X_2 \\ X_2 - X_1 & X_1 < X_2. \end{cases} \tag{3}$$

Since the two memory addresses to define the memory region are picked up independently, the PDF of the memory region to be verified is the product of the PDF function for $X_1$ and $X_2$ from Eq. (1):

$$F(X_1, X_2) = F(X_1) * F(X_2) = \begin{cases} \frac{1}{L^2} & X_1, X_2 \in [0, L] \\ 0 & \text{Otherwise.} \end{cases} \tag{4}$$

Therefore, the expected value $\mathbb{E}(Y) = \mathbb{E}(g(X_1, X_2))$ is given by

$$\begin{aligned} \mathbb{E}(Y) &= \int_0^L \int_0^L g(x_1, x_2) F(x_1, x_2) dx_2 dx_1 \\ &= \frac{1}{L^2} \int_0^L \int_0^L |x_1 - x_2| dx_2 dx_1 \\ &= \frac{1}{L^2} \int_0^L \int_0^{X_1} (x_1 - x_2) dx_2 dx_1 + \frac{1}{L^2} \int_0^L \int_{X_1}^L (x_2 - x_1) dx_2 dx_1 \\ &= \frac{L^3}{6L^2} + \frac{L^3}{6L^2} = 1/3 * L \end{aligned} \tag{5}$$

□

One of the potential attacks against the memory randomization in MTRA is that an adversary attempts to capture a

large number of attestation responses to retrieve the complete firmware. We further evaluate the probability that an adversary is able to retrieve the entire memory regions in terms of the number of attestation responses captured. We assume that adversary is able to capture the combinations of two addresses (beginning address and end address) from *all* the attestations.

**Lemma 6.2.** *If the memory regions to be verified are randomly and independently chosen and the memory address starts from 0 to long integer L−1, the probability in exact N attestations to cover the entire memory region (f(N)) ,whose size is L , is*

$$f(N) = \sum_{i=1}^{N} (-1)^{i+1} \left(\frac{2}{L(L-1)}\right)^{i} \tag{6}$$

**Proof.** We can use the mathematical induction to prove this. When $N = 1$, the only combination of start and ending memory addresses to cover the entire region is 0 and $L-1$. The probability of this combination is $C_L^2$. Therefore,

$$f(1) = \frac{1}{C_L^2} = \frac{2}{L(L-1)} \tag{7}$$

It matches with Eq, (6) for $N = 1$.

Now we assume that Eq. (6) holds for $N$ attestations. The probability for the $(N+1)$th attestation to verify the entire memory region is the probability of the previous $N$ attestations failing to cover the entire memory region and the probability that the $N+1$th attestation is able to cover the starting and ending memory addresses for the entire memory region. Therefore, $f(N+1)$ is

$$\begin{aligned} f(N+1) &= (1 - f(N)) * f(1) \\ &= f(1) + (-1) * f(N) * f(1) \\ &= f(1) + \sum_{i=2}^{N+1} (-1)^{i+1} \left(\frac{2}{L(L-1)}\right)^{i} \\ &= \sum_{i=1}^{N+1} (-1)^{i+1} \left(\frac{2}{L(L-1)}\right)^{i} \end{aligned} \tag{8}$$

Therefore, $f(N+1)$ holds for Eq. (6). This lemma holds for $\forall N \in \mathbb{Z}+$. □

Based on Lemma 6.2, we can have the following lemma:

**Lemma 6.3.** *If the memory regions to be verified are randomly and independently chosen and the memory address starts from 0 to long integer L − 1, the cumulative probability to cover the entire memory region after N attestations (T(N)) is*

$$T(N) = f(1) \sum_{i=1}^{N} i(-f(1))^{N-i} \tag{9}$$

*where f(1) is defined in Lemma 6.2:*

$$f(1) = \frac{2}{L(L-1)},$$

**Proof.** We can use the mathematical induction to prove this. When $N = 1$, the cumulative probability of the first attestation to cover the entire memory region is the same as $F(1)$ in Eq. (7) from Lemma 6.2:

$$T(1) = f(1)$$

It matches with Eq. (9) for $N = 1$.

Now we assume that Eq. (9) holds for $N$ attestations. The cumulative probability for the $(N+1)$th attestation to verify the entire memory region is

$$\begin{aligned} T(N+1) &= T(N) + f(N+1) \\ &= f(1) \sum_{i=1}^{N} i(-f(1))^{N-i} \\ &\quad + \sum_{i=1}^{N+1} (-1)^{i+1} (f(1))^{i} \\ &= f(1) \left( \sum_{i=1}^{N} (N-i)(-f(1))^{i} \right. \\ &\quad \left. + \left( \sum_{i=1}^{N} (-f(1))^{i} + 1 \right) \right) \\ &= f(1) \left( \sum_{i=1}^{N} (N-i+1)(-f(1))^{i} + 1 \right) \\ &= f(1) \left( \sum_{i=1}^{N+1} (i)(-f(1))^{N-i+1} \right) \end{aligned}$$

So Eq. (9) holds for $N+1$. This lemma holds for $\forall N \in \mathbb{Z}+$. □

**Theorem 6.4.** *If the memory regions to be verified are randomly and independently chosen and the memory address starts from 0 to long integer L − 1, the cumulative probability to verify the entire memory region only depends on the size of the memory region (i.e., L) when the number of attestations is infinite. The probability (P):*

$$P = \frac{2}{L(L-1)+2}$$

*where L > 2.*

**Proof.** Eq. (6) can be organized as follows:

$$f(N) = \sum_{i=0}^{\lceil N/2 \rceil} \left(\frac{2}{L(L-1)}\right)^{2i+1} - \sum_{i=1}^{\lceil N/2 \rceil} \left(\frac{2}{L(L-1)}\right)^{2i} \tag{10}$$

As the first and second terms are both the summations of geometry series and the common ratio ,$\left(\frac{2}{L*(L-1)}\right)^2 < 1$ if $L > 2$. There is a limit when $N$ (i.e., number of attestations) becomes infinite:

$$\begin{aligned} lim_{N\to\infty} f(N) &= lim_{N\to\infty} \sum_{i=0}^{\lceil N/2 \rceil} \left(\frac{2}{L(L-1)}\right)^{2i+1} \\ &\quad - lim_{N\to\infty} \sum_{i=2}^{\lceil N/2 \rceil} \left(\frac{2}{L(L-1)}\right)^{2i} \\ &= \frac{\frac{2}{L(L-1)}}{\left(1 - \frac{2}{L(L-1)}\right)^2} - \frac{\left(\frac{2}{L(L-1)}\right)^2}{(1 - \frac{2}{L(L-1)})^2} \\ &= \frac{2}{L(L-1)+2} \end{aligned} \tag{11}$$

□

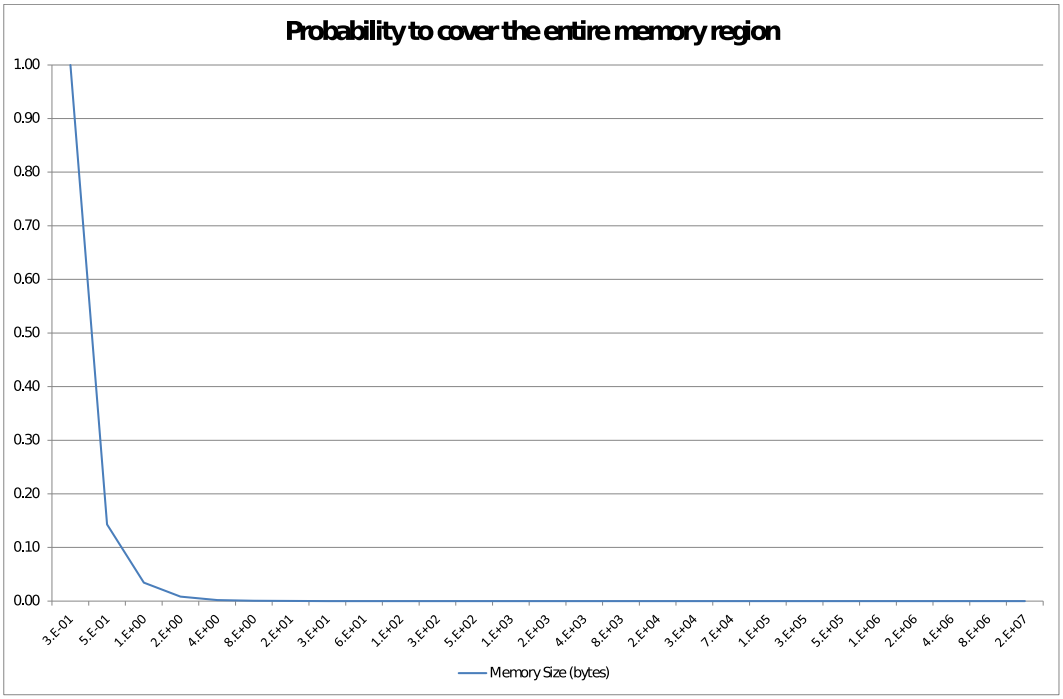



**Fig. 12 – Probability to cover the entire memory region through attestation memory randomization: Adversary is not able to retrieve the contents of the entire memory region even though it captures a number of the attestation responses.**

**Table 2 – Comparison of MTRA, SCUBA (Seshadri et al., 2006), SEDA (Asokan et al., 2015), PIV (Agrawal et al., 2015), TRAP (Tan et al., 2011) and SMARM (Carpent et al., 2018a).**

| | MTRA | SCUBA | SEDA | PIV | TRAP | SMARM |
|---|---|---|---|---|---|---|
| TPM enabled | Partial | No | No | Partial | Yes | No |
| Synchronisation required? | No | Yes | No | No | No | No |
| Defends against TOCTTOU attacks? | Yes | No | Yes | No | Yes | Yes |
| Defends against wormhole attacks? | Yes | No | No | No | Yes | No |
| Defends against MIM? | Yes | No | Yes | No | Yes | No |
| Defends against Verifier-based DoS attacks? | Yes | Yes | No | Yes | No | Yes |
| Communication overhead (bytes) | 60+ | 160+ | 512+ | 60+ | 256+ | 1,000+ |

Fig. 12 shows the probability to cover the entire memory regions during memory randomization in relation to the memory size (i.e., Eq. (11)). If the memory size is greater than 8 bytes, the probability to retrieve the entire memory regions by capturing a number of attestation responses is negligible.

### 6.7. *Comparison with existing attestation protocols*

Table 2 shows the differences between MTRA (the protocol in this paper), SCUBA (Seshadri et al., 2006), SEDA (Asokan et al., 2015), TRAP (Tan et al., 2011) and SMARM (Carpent et al., 2018a). We implemented MTRA with the TPM emulator (Strasser and Stamer, 2008) on powerful devices (i.e., Odroid XU4), which are the verifiers after they pass the attestation. MTRA does not require strict timing on the attestation process, while SCUBA requires synchronisation between the verifier and prover. Compared to SEDA (Asokan et al., 2015), MTRA can defend against wormhole attacks as we employed a zone-based one-way hash chain to prevent the attestation response being replayed in a different network area where the firmware has not been attested. Compared to our earlier system, TRAP (Tan et al., 2011), this protocol is more cost-efficient and adaptive, as not all the devices in the network are required to be TPM-enabled. Our protocol not only reduces the deployment costs but is also robust against verifier-based DoS attacks, as the verifiers must perform more tasks than the provers to formalize an attestation. On the other hand, TRAP was not able to address verifier-based DoS attacks as all the devices run TPM to process the attestation challenges, which consumes more resources. The communication overhead required to run each protocol is universal so we also include the comparison results in Table 2. MTRA does not send a digital signature for attestation verification while SEDA and TRAP do. SCUBA must perform synchronisation between the verifier and prover so it requires more communication overhead than MTRA. PIV (Agrawal et al., 2015) has a communication overhead similar to MTRA, but it can only be applied in one-hop networks. Therefore, PIV does not consider wormhole attacks and MIM attacks.

The recent remote attestation, SMARM (Carpent et al., 2018a), can operation as usual when remote attestation is performed. However, it did not considered the network-based attack such as wormhole attack and its communication overhead is the scale of mega-bytes while MTRA's communication overhead is in kilo-bytes.

## 7. Conclusions

We have proposed a remote attestation protocol for IoT networks (MTRA) in this paper. In MTRA, we tailored separate schemes for both TPM-enabled IoT devices and IoT devices that could not run TPMs. We discussed potential attacks against our scheme and proposed countermeasures. We implemented MTRA on two single-board computers, Odroid-XU4 for the TPM-enabled devices and Raspberry Pi for the non-TPM devices, as well as the attested memory randomization. The performance evaluation indicates MTRA is cost-efficient and is more suitable for resource-constrained devices than existing remote attestation protocols. In future works, we will investigate the algorithms to further randomized the memory regions to be attested and increase the expected memory size in MTRA.

## Acknowledgment

This research was supported partially by the Australian Government through the Australian Research Council's Discovery Projects funding scheme (project DP150100564). The views expressed herein are those of the authors and are not necessarily those of the Australian Government or Australian Research Council.

## Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.cose.2018.10.008

## REFERENCES

Abbasi AA, Younis M. A survey on clustering algorithms for wireless sensor networks. Comput Commun 2007;30(14–15):2826–41. doi:10.1016/j.comcom.2007.05.024.

Abera T, Asokan N, Davi L, Ekberg JE, Nyman T, Paverd A, Sadeghi AR, Tsudik G. C-FLAT: control-Flow Attestation for embedded systems software, 2016. arXiv: 1605.07763.

Agrawal S, Das M, Mathuria A, Srivastava S. Program integrity verification for detecting node capture attack in wireless sensor network. LNCS, 9478. Springer International Publishing; 2015. p. 419–40. doi:101007/978-3-319-26961-0_25.

Alqassem I, Svetinovic D. A taxonomy of security and privacy requirements for the internet of things (IOT). In: Proceedings of the 2014 IEEE international conference on industrial engineering and engineering management; 2014. p. 1244–8. doi:10.1109/IEEM.2014.7058837.

Anderson M. An a for raspberry pi b+, IEEE Spectrum, Tech talk Blog, July 25th, 2014. URL http://spectrum.ieee.org/tech-talk/geek-life/hands-on/an-a-for-raspberry-pi-b.

Asokan N, Brasser F, Ibrahim A, Sadeghi AR, Schunter M, Tsudik G, Wachsmann C. SEDA: scalable embedded device attestation. In: Proceedings of CCS'15. New York, NY, USA: ACM; 2015. p. 964–75. doi:10.1145/2810103.2813670.

Brasser F, Rasmussen KB, Sadeghi AR, Tsudik G. Remote attestation for low-end embedded devices: the prover's perspective. Proceedings of DAC'16. New York, NY, USA: ACM, 2016. doi: 10.1145/2897937.2898083.

Carpent X, Rattanavipanon N, Tsudik G. Remote attestation of IOT devices via SMARM: shuffled measurements against roving malware. In: Proceedings of the HOST. IEEE Computer Society; 2018a. p. 9–16. doi: 10.23919/DATE.2018.8342195.

Carpent X, Tsudik G, Rattanavipanon N. ERASMUS: efficient remote attestation via self-measurement for unattended settings. In: Proceedings of the design, automation & test in Europe conference & exhibition, DATE 2018, Dresden, Germany, March 19–23, 2018; 2018b. p. 1191–4. doi:10.23919/DATE.2018.8342195.

Challa S, Wazid M, Das AK, Kumar N, Reddy AG, Yoon E, Yoo K. Secure signature-based authenticated key establishment scheme for future IOT applications. IEEE Access 2017;5:3028–43. doi:10.1109/ACCESS.2017.2676119.

Feng W, Qin Y, Zhao S, Feng D. AAOT: Lightweight attestation and authentication of low-resource things in IOT and CPS.. Comput Netw 2018;134:167–82. URL http://dblp.uni-trier.de/db/journals/cn/cn134.html#FengQZF18.

Group T.C.. Trusted computing group - trusted platform module (TPM) summary. http://www.trustedcomputinggroup.org/resource/trusted-platform-module-2-0-a-brief-introduction; last access on 29/01/2016.

Hu Y-C, Jakobsson M, Perrig A. Efficient constructions for One-Way hash chains. LNCS, 3531. Berlin Heidelberg: Springer; 2005. p. 423–41. doi:101007/11496137_29.

Hu Y-c, Perrig A, Johnson DB. Wormhole attacks in wireless networks. In: Proceedings of the IEEE JSAC, Volume 24, No. 2; 2006. p. 370–80. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.117.698.

Kil C, Sezer EC, Azab AM, Ning P, Zhang X. Remote attestation to dynamic system properties: towards providing complete system integrity evidence. In: Proceedings of the IEEE/IFIP ICDSN'09. IEEE; 2009. p. 115–24. doi:10.1109/dsn.2009.5270348.

Klein G, Elphinstone K, Heiser G, Andronick J, Cock D, Derrin P, Elkaduwe D, Engelhardt K, Kolanski R, Norrish M, Sewell T, Tuch H, Winwood S. sel4: Formal verification of an os kernel. In: Proceedings of the ACM SIGOPS 22nd symposium on operating systems principles. New York, NY, USA: ACM; SOSP '09; 2009. p. 207–20. doi:10.1145/1629575.1629596.

Krauss C, Stumpf F, Eckert C. Detecting node compromise in hybrid wireless sensor networks using attestation techniques. In: Proceedings of ESAS '07. Berlin, Heidelberg: Springer-Verlag; 2007. p. 203–17. URL http://portal.acm.org/citation.cfm?id=1784425.

NS3: discrete-event network simulator, https://www.nsnam.org/, [Accessed on 18 April 2017]; 2011.

Seshadri A, Luk M, Perrig A, van Doorn L, Khosla P. SCUBA: secure code update by attestation in sensor networks. In: Proceedings of WISE '06. New York, NY, USA: ACM; 2006. p. 85–94. doi:10.1145/1161289.1161306.

Seshadri A, Perrig A, van Doorn L, Khosla P. SWATT: software-based attestation for embedded devices. In: Proceedings of IEEE symposium on security and privacy; 2004. p. 272–82. URL http://www.netsec.ethz.ch/publications/papers/swatt.pdf.

Song K, Seo D, Park H, Lee H, Perrig A. OMAP: one-Way memory attestation protocol for smart meters. In: Proceedings of the 2011 ISPAW. IEEE; 2011. p. 111–18. doi:10.1109/ispaw.2011.37.

Strasser M, Stamer H. A software-based trusted platform module emulator. In: Proceedings of Trust' 08. Berlin, Heidelberg: Springer-Verlag; 2008. p. 33–47. doi:10.1007/978-3-540-68979-9_3.

Tan H, Hu W, Jha S. A TPM-enabled remote attestation protocol (TRAP) in wireless sensor networks. In: Proceedings of the PM2HW2N '11. New York, NY, USA: ACM; 2011. p. 9–16. doi:10.1145/2069087.2069090.

Tan H, Tsudik G, Jha S. MTRA: multiple-tier remote attestation in IOT networks. In: Proceedings of the 2017 IEEE conference on communications and network security (CNS); 2017. p. 1–9. doi:10.1109/CNS.2017.8228638.

The legion of bouncy castle [online], 2009, http://www.bouncycastle.org, [Accessed 16 September 2016].

Trousers: The open-source TCG software stack, 2008, http://trousers.sourceforge.net/ Trust '08, [Accessed 13 September 2016]. 2008

US Secure Hash Algorithm 1 (SHA1), https://tools.ietf.org/html/rfc3174, [Accessed 16 September 2016]; 2001.

Wang J, Hong Z, Zhang Y, Jin Y. Enabling security-enhanced attestation with intel SGX for remote terminal and iot. IEEE Trans Integr Circuits Syst 2018;37(1):88–96.

Wazid M, Das AK, Odelu V, Kumar N, Conti M, Jo M. Design of secure user authenticated key management protocol for generic IOT networks. IEEE Internet of Things J 2018;5(1):269–82. doi:10.1109/JIOT.2017.2780232.

**Hailun Tan** is a cyber-security consultant in R&D department of NOJA PowerSwitchgear Pty Ltd since 2017. He obtained his Ph.D. degree from the University of New South Wales (UNSW)in 2010. Before coming to NOJA Power Switchgear Pty Ltd., He had worked as researcher in the Australia's Information and Communications Technology Research Centre of Excellence (NICTA) (2006–2011) and The Commonwealth Scientific and Industrial Research Organisation (CSIRO) (2011–2013). From 2013 to 2015, he worked as a cyber-security consultant in Australian Federal Government. From 2016 to 2017, he was a postdoc fellow and lecturer in UNSW, focusing on security in IoT. His main research interests are in security protocol design in wireless networks and embedded systems.

**Gene Tsudik** is a Chancellor's Professor of Computer Science at the University of California, Irvine (UCI). He obtained his Ph.D. in Computer Science from USC in 1991. Before coming to UCI in 2000, he was at IBM Zurich Research Laboratory (1991–1996) and USC/ISI (1996–2000). His research interests include many topics in security, privacy and applied cryptography. Gene Tsudik is a Fulbright Scholar, Fulbright Specialist (twice), a fellow of ACM, a fellow of IEEE, a fellow of AAAS, and a foreign member of Academia Europaea. From 2009 to 2015 he served as Editor-in-Chief of ACM Transactions on Information and Systems Security (TISSEC, renamed to TOPS in 2016). Gene was the recipient of 2017 ACM SIGSAC Outstanding Contribution Award. He is also the author of the first crypto-poem published as a refereed paper.

**Sanjay K. Jha** is a Professor and Head of the Networked Systems and Security Group and Director of Cybersecurity and Privacy Lab at the School of Computer Science and Engineering at the University of New South Wales. His research activities cover a wide range of topics in networking including Wireless Sensor Networks, Ad-hoc/Community wireless networks, Resilience and Multicasting in IP Networks and Security protocols for wired/wireless networks. Sanjay has published over 200 articles in high quality journals and conferences. He is the principal author of the book Engineering Internet QoS and a co-editor of the book Wireless Sensor Networks: A Systems Perspective. He served as editor of the IEEE Transactions on Mobile Computing (TMC). He currently serves as editor of the IEEE Transactions on Dependable and Secure Computing (TDSC).