2018 IEEE 37th International Symposium on Reliable Distributed Systems

US-AID: Unattended Scalable Attestation of IoT Devices

Ahmad Ibrahim
TU Darmstadt, Darmstadt, Germany
Ahmad.Ibrahim@trust.tu-darmstadt.de

Ahmad-Reza Sadeghi
TU Darmstad, Darmstadt, Germany
Ahmad.Sadeghi@trust.tu-darmstadt.de

Gene Tsudik
UC Irvine, Irvine, CA, USA
Gene.Tsudik@uci.edu

***Abstract*—Embedded devices, personal gadgets and networks thereof are becoming increasingly pervasive, mainly due the advent of, and hype surrounding, the so-called Internet of Things (IoT). Such devices often perform critical actuation tasks, as well as collect, store and process sensitive data. Therefore, as confirmed by recent examples (such as the Mirai botnet), they also represent very attractive attack targets. To mitigate attacks, remote attestation (RA) has emerged as a distinct security service that aims at detecting malware presence on an embedded device. Most prior RA schemes focus on attesting a single device and do not scale. In recent years, schemes for collective (group or swarm) RA have been designed. However, none is applicable to autonomous and dynamic network settings.**

This paper presents US-AID – the first collective attestation schemes for large autonomous dynamic networks of embedded devices. US-AID verifies overall network integrity by combining continuous in-network attestation with a key exchange mechanism and Proofs-of-non-Absence. Using device absence detection US-AID defends against physical attacks that require disconnecting attacked devices from the network for a non-negligible time.

We demonstrate feasibility of US-AID via proof-of-concept implementations on a state-of-the-art security architectures for low-end embedded devices and on an autonomous testbed comprising six drones. We also assess its scalability and practicality via extensive simulations.

***Keywords*-Security; Attesation; IoT; Embedded Devices;**

I. Introduction

In recent years, embedded devices proliferated into numerous domains, collecting, processing and exchanging sensitive information, while performing safety- and security-critical operations. Interconnection of embedded devices within the existing network infrastructure enables a wide range of applications, including centrally managed "smart" environments (e.g., cars, homes, buildings, and factories) and autonomous dynamic ad-hoc networks, such as vehicular ad-hoc networks for autonomous driving, and swarms of self-organizing collaborating embedded devices, e.g., robots and drones. Despite many benefits, allowing embedded devices to be accessed and controlled remotely poses a formidable security challenge, since it greatly broadens the attack surface and magnifies consequences of a successful attack.

Remote malware attacks usually involve modifying device firmware and/or software, e.g., Stuxnet [28], and Jeep hack [21]. Unfortunately, sophisticated security features common on general-purpose computing devices are lacking on most embedded devices, due to their limited resources. This makes them attractive targets for remote attackers. On the other hand, IoT networks can involve hundreds and even thousands of autonomous, mobile and inter-connected devices, operating in hostile environments. In particular, whenever devices are not always within a physical security perimeter, they can be subject to capture and physical attacks.

For this reason, much effort has been invested in recent years into *Remote Attestation (RA)* – a distinct security service that aims to detect presence of malware on a remote and potentially compromised embedded device. RA is usually realized as a protocol between an untrusted *Prover* device that securely reports its current software state to a trusted remote *Verifier*. Most prior work focused on this setting. There are three general types of RA protocols: *software-based* [22], [18], *hardware-based* [2], [17], and hybrid (based on software/hardware co-design) *hybrid* [10], [11], [16]. Currently, hybrid attestation is viewed as the most promising approach, since it provides strong security guarantees while involving lightweight hardware features: a small amount of Read-Only Memory (ROM), and a simple Memory Protection Unit (MPU).

Most prior attestation schemes – regardless of the type – are geared for a single-prover setting, i.e., they do not scale to many provers, and only consider remote software-only attacks. Several recent efforts [5], [1], [13], [6] yielded attestation methods for networks or groups of devices that provide *collective* or *swarm attestation*. SEDA [5] efficiently attests a network of low-end embedded devices. SANA [1] is based on Optimistic Aggregate Signatures (OAS) to provide better efficiency and resiliency. Finally, DARPA [13] allows detection of physical attacks by extending SEDA with a periodic network-wide heartbeat protocol. DARPA assumes that a physical attack requires disconnecting the target device for a non-negligible amount of time, and uses a global absence detection protocol to detect such physical attacks.

Problem Statement. Prior collective RA schemes focused on *centralized* IoT networks with *restricted mobility*, meaning that network topology must remain static during the entire RA protocol execution. Moreover, such schemes cannot mitigate *physical attacks*, except for DARPA [13], which scales poorly, since it incurs high computation and communication costs – quadratic in network size. Some emerging IoT networks have highly dynamic topologies and include large numbers of

2575-8462/18/$31.00 ©2018 IEEE
DOI 10.1109/SRDS.2018.00013

devices. In such settings, multiple devices can be subject to physical attacks.

Goals and Contributions. This paper presents US-AID – the first attestation protocol for *large, autonomous and dynamic-topology* networks of embedded devices. Designing US-AID poses several challenges. In particular, attestation and physical attack detection should be combined with key management in order to identify malicious devices, while preserving scalability (i.e., keeping overhead constant in terms of network size) and device mobility. This requires ensuring secure flow of attestation results throughout the network, in order to allow mobile devices to prove their trustworthiness to new neighbors while incurring minimal and constant costs. Finally, computation, communication, and energy costs should be evenly distributed over all devices, to prevent performance bottlenecks. US-AID combines continuous attestation of neighbors and periodic local heartbeats with a key exchange mechanism to detect both software attacks and physical attacks with minimal cost. It handles dynamic topology using a dedicated roaming protocol.

Overall, this paper makes three technical contributions:

- *In-network Attestation:* US-AID is the first efficient, scalable and secure attestation scheme for large autonomous dynamic networks of embedded devices, that allows isolation of potentially malicious devices.
- *Proof-of-Concept Implementation:* US-AID was implemented on two recent RA architectures for low-end embedded devices: SMART [10] and TrustLite [16], and on an autonomous testbed formed of six interconnected drones equipped with a Raspberry Pi 3 Model B.
- *Performance Analysis:* We conducted a thorough performance and security assessment of US-AID with simulation results for networks of up to $1,000,000$ devices.

II. US-AID

A. System Model

We consider a dynamic network $\mathcal{E}$ composed of devices: E_i for $i = 1, \ldots, n$. Each E_i has a unique ID id_i. Devices can be heterogeneous and might be spread over a large physical area. As discussed in Section II-B, devices can have various software and hardware. However, all devices should support a lightweight RA architecture. We assume that mobility is contiguous, i.e., a device can move gradually and change its set of direct neighbors.[1] However, a moving device never disappears from one neighborhood and instantaneously reappears in a remote neighborhood. Each E_i is always available and reachable, e.g., smart IoT environments. In other words, we do not consider networks where devices can be completely switched off for an extended period of time.[2] We denote by $\mathcal{ID}_i$ the set of current neighbors of E_i. The (trusted) network operator O is responsible for initialization and maintenance of $\mathcal{E}$, e.g., adding new, and removing defective, devices. The overall goal of US-AID is to detect and isolate devices as described in our adversary model in Section II-B). Policies that govern the network reaction to isolated malicious devices are out-of-scope.

[1]Two devices are considered neighbors if they can communicate directly, i.e., if they are within each other's communication range.

[2]Sleep mode is allowed as long as devices are awake to execute the protocol.

B. Requirements Analysis

Adversary and Trust Model. We assume that O is fully trusted and initializes devices in a secure environment. Other entities (devices) can potentially be under the control of $\mathcal{A}$, which aims to compromise devices and evade detection, similar to Stuxnet [28]. Since $\mathcal{A}$ aims to remain stealthy, we do not consider DoS attacks which reveal $\mathcal{A}$'s presence.

$\mathcal{A}$ can compromise communication channels and/or devices. More concretely, it can: (1) control all communication channels, inject its own packets and eavesdrop on, modify, and delay packets exchanged between devices; (2) exploit software vulnerabilities to gain control over devices, extract unprotected secrets and modify unprotected software; and (3) mount physical attacks in order to modify devices' hardware or software components and/or learn hardware-protected secrets. In order to physically attack a device, we assume that $\mathcal{A}$ has to turn it off for a non-negligible amount of time t_{phy}.

We consider three types of physical attacks: *invasive*, *semi-invasive* and *non-invasive*. An invasive attack [24] involves $\mathcal{A}$ trying to extract information from a device through direct access to its internal components. It requires sophisticated and expensive lab equipment. Semi-invasive attacks [25] are less expensive and less complicated. They use cheaper equipment (e.g., laser microscopes) and only require de-capsulation. Examples include: thermal imaging, ultra-violet attacks, optical fault injections and laser scanning. Both invasive and semi-invasive attacks require possession of the victim device for a non-negligible period of time: from hours to weeks [24], [23]. Meanwhile, non-invasive attacks [29] use low-cost electrical engineering tools, such as fault injection and various *side-channels* (e.g., time, power or electromagnetic radiation) to extract device's cryptographic keys during normal operation.

We do not claim that US-AID can mitigate all physical attacks. We specifically exclude non-invasive hardware side-channel attacks that might not require physically disconnecting the target device from the network. presence and proximity, unlike software attacks where $\mathcal{A}$ remotely and automatically infects many devices. Since physical attacks are usually not scalable, we assume that $\mathcal{A}$ can physically attack a limited number of devices.

Main objectives. In the assumed settings, there is no central authority to initiate the RA protocol and verify results. Hence, we envisage a collective attestation scheme that: (1) is efficient, ideally with constant computation and communication costs in terms of tnetwork size, achieved by requiring each device to only assess trustworthiness of its neighbors; (2) assures detection and isolation of compromised devices, achieved by combining attestation and absence detection with key exchange; and (3) captures network's dynamic behavior, whereby devices move and gradually change their neighborhood, done through a dedicated roaming protocol which allows the secure flow of RA results.

Device Requirements. Every E_i has a signing key-pair (sk_i, pk_i), and a lightweight RA trust anchor. The trust anchor may consist of a small ROM and a *simple* MPU, as described in Section IV. This hardware protects US-AID protocol code and cryptographic keys against software-based attacks. We assume that each E_i has a *loosely* synchronized reliable read-only clock (RROC), i.e., a clock that is not modifiable by software. We acknowledge that tight clock synchronization across all devices is not always feasible, especially, in very large networks. For this reason, US-AID requires loosely synchronized clocks with skews on the order of minutes.

Network Requirements. We assume that the network is always connected and devices are available at all times. Time to transmit a message between two neighboring devices is upper-bounded by t_{tr}. We assume smooth mobility, meaning that devices approach each other gradually, i.e., when E_i moves towards E_j it establishes a neighborhood with all devices on its path to E_j. Consequently, before E_i and E_j come within each others range, they have at least one common neighbor.[3]

C. High Level Protocol Description

We present a high-level overview of US-AID based on the sample network $\mathcal{E}$ illustrated in Figure 1 with E_1–E_8. US-AID incorporates multiple protocols executed at different phases and for various purposes.

- init ①: is executed by O before enrollment. O initializes each device with the required cryptographic materials, e.g., a signing key-pair and a device certificate (E_8 in the figure).
- connect: is executed when two devices become neighbors, i.e., when a new device (E_8 ②) joins $\mathcal{E}$, or changes its position (E_6 ③). The purpose of connect is twofold: (1) it allows devices to share secret keys used for authenticating protocol messages; and (2) it enables benign devices to report their state to newly established neighbors. New neighbors establish a secure communication (i.e., share keys) only if they can verify each other's trustworthiness.
- attest ④: is launched at random times between every two neighboring devices, allowing each device to attest its direct neighbors and keep track of their software state. For instance, E_6 attests its every neighbor, e.g., E_5. Then, E_6 records its list of neighbors that attested successfully, and drops secure communication (i.e., deletes shared keys) with all devices that failed attestation.
- beat ⑤: is executed periodically allowing each device to keep track of its neighbors' presence, based on absence detection. Each device sends a heartbeat to all neighbors demonstrating its presence. Each device records the list of devices from which it received a heartbeat and drops secure communication with all devices that did not send a heartbeat. Moreover, as a response to a heartbeat, each device gets Proof-of-non-Absence (PonA) tokens from its neighbors. These are used by a mobile device to prove its trustworthiness to new neighbors through connect.

Each device keeps track of software/hardware trustworthiness of its neighbors through attestation and absence detection. By establishing secure communication with only those neighbors that have proven their trustworthiness, US-AID allows formation of a secure connected sub-network of devices that are not compromised, i.e., passed attestation and are always present, e.g., for E_6 this includes: $\{E_1, E_3, E_4, E_5, \text{and } E_7\}$. This allows detecting and isolating possibly compromised devices (E_2).

[3]Network partitioning is tolerated, though it requires O to reconnect partitions. Similarly, power losses and node crashes always require O's assistance, e.g., to replace a depleted battery.

III. Detailed Protocol Description

A. Notation and Definition

Before describing US-AID, we introduce the notation:

Benign and Secure Device Lists. Each E_i keeps two lists: (1) $\mathcal{B}_i$ stores IDs of neighbors, that E_i believes to have benign software. They are known to E_i through continuous attestation. (2) $\mathcal{S}_i$ stores IDs of neighbors that E_i believes have not been physically attacked. These are known to E_i through absence detection, as described in Section II-B).

Heartbeat Interval. Time is divided into uniform-size heartbeat intervals. An interval is upper-bounded by the minimum time for $\mathcal{A}$ to perform a physical attack: $t_{hb} < t_{phy}$. Every heartbeat interval starts at T_{hb} and is identified by its ID q_i.

Present Device List. For every heartbeat interval, each E_i keeps a list $\mathcal{P}_i$ of neighbors present during this interval.

Heartbeat. Each E_i with ID id_i generates at T_{hb} of every heartbeat interval q_i a heartbeat. E_i authenticates the heartbeat using a MAC μ_{ij} based on a heartbeat key k_{ij}^{beat} shared between E_i and every neighbor E_j, and sends it to E_j. We denote this heartbeat by $HB_{ij} = \{\{q_i, id_i\}, \mu_{ij}\}$. It proves continuous presence of E_i to E_j.

Proof-of-Secure-Enrollment (PoSE). At E_i's initialization time T_{init}, O generates a token formed of E_i's ID id_i and T_{init}, creates a digital signature σ_i over it, and sends it to E_i. We denote this token by PoSE $\pi_i = \{\{id_i, T_{\mathsf{init}}\}, \sigma_i\}$. π_i proves that E_i was enrolled securely.

Proof-of-non-Absence (PonA). Every E_i with ID id_i proves its presence, at heartbeat interval q_i, to all its neighbors. As a response, each neighbor E_j creates a set of tokens formed of id_i, q_i, and time $T_A^{E_i}$ of the last attestation of E_i by E_j. It then authenticates them based on a heartbeat key k_{jk}^{beat} shared with every neighbor E_k of E_j, and sends them to E_i. We denote by $\psi_{jk/i} = \{\{id_i, q_i, T_A^{E_i}\}, \hat{\mu}_{jk/i}, id_k\}$ an individual token, and by Π_i – the set of all authenticated tokens received by E_i from every neighbor E_j. Each $\psi_{jk/i}$ in Π_i proves that E_i was continuously present and successfully attested.

B. Protocol Details

US-AID includes the following protocols:

Initialization. Each E_i is initialized by O with:

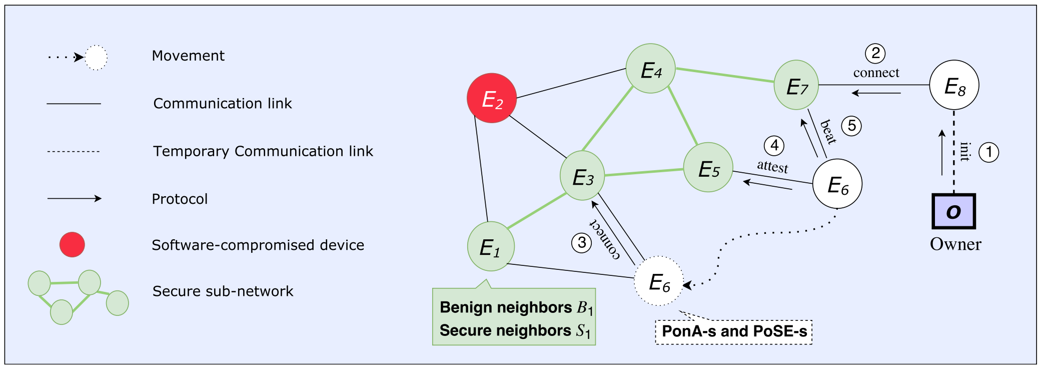
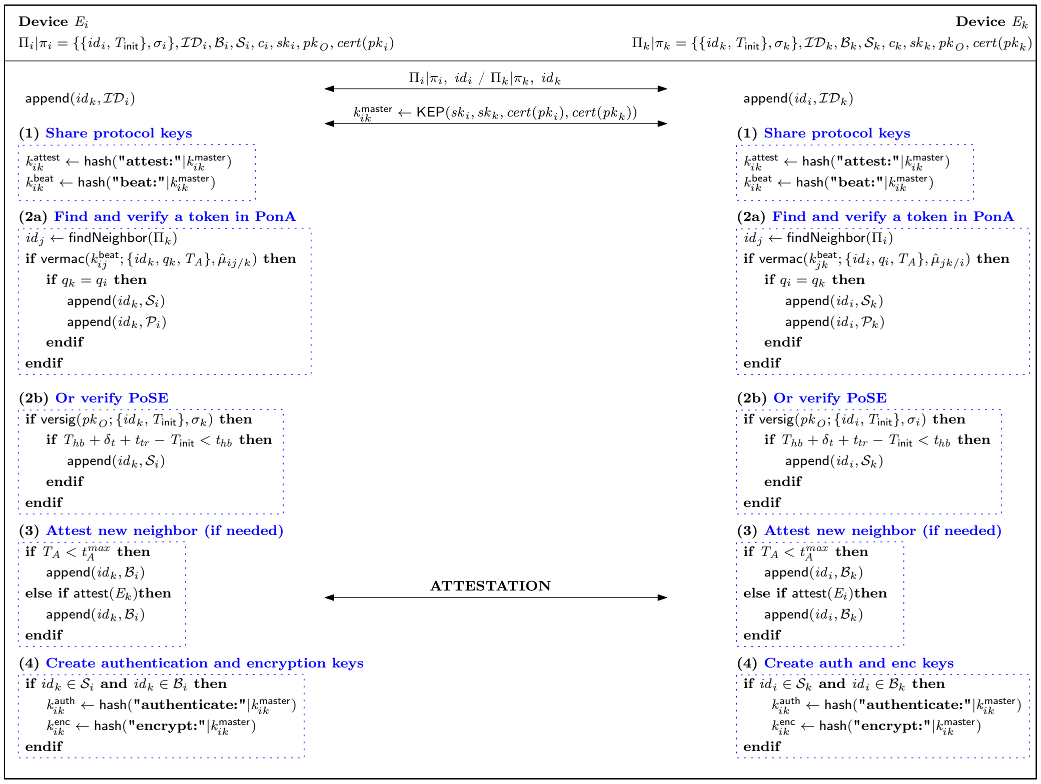


Figure 1: US-AID in a network of 8 devices.



Device E_i

$\Pi_i|\pi_i = \{\{id_i, T_{\text{init}}\}, \sigma_i\}, \mathcal{ID}_i, \mathcal{B}_i, \mathcal{S}_i, c_i, sk_i, pk_O, cert(pk_i)$

Device E_k

$\Pi_k|\pi_k = \{\{id_k, T_{\text{init}}\}, \sigma_k\}, \mathcal{ID}_k, \mathcal{B}_k, \mathcal{S}_k, c_k, sk_k, pk_O, cert(pk_k)$

$\xleftrightarrow{\Pi_i|\pi_i,\ id_i\ /\ \Pi_k|\pi_k,\ id_k}$

$\xleftrightarrow{k_{ik}^{\text{master}} \leftarrow \text{KEP}(sk_i, sk_k, cert(pk_i), cert(pk_k))}$

$\xleftrightarrow{\textbf{ATTESTATION}}$

Device E_i:

```
append($id_k, \mathcal{ID}_i$)

(1) Share protocol keys
$k_{ik}^{\text{attest}} \leftarrow$ hash("attest:"|$k_{ik}^{\text{master}}$)
$k_{ik}^{\text{beat}} \leftarrow$ hash("beat:"|$k_{ik}^{\text{master}}$)

(2a) Find and verify a token in PonA
$id_j \leftarrow$ findNeighbor($\Pi_k$)
if vermac($k_{ij}^{\text{beat}}; \{id_k, q_k, T_A\}, \hat{\mu}_{ij/k}$) then
    if $q_k = q_i$ then
        append($id_k, \mathcal{S}_i$)
        append($id_k, \mathcal{P}_i$)
    endif
endif

(2b) Or verify PoSE
if versig($pk_O; \{id_k, T_{\text{init}}\}, \sigma_k$) then
    if $T_{hb} + \delta_t + t_{tr} - T_{\text{init}} < t_{hb}$ then
        append($id_k, \mathcal{S}_i$)
    endif
endif

(3) Attest new neighbor (if needed)
if $T_A < t_A^{max}$ then
    append($id_k, \mathcal{B}_i$)
else if attest($E_k$)then
    append($id_k, \mathcal{B}_i$)
endif

(4) Create authentication and encryption keys
if $id_k \in \mathcal{S}_i$ and $id_k \in \mathcal{B}_i$ then
    $k_{ik}^{\text{auth}} \leftarrow$ hash("authenticate:"|$k_{ik}^{\text{master}}$)
    $k_{ik}^{\text{enc}} \leftarrow$ hash("encrypt:"|$k_{ik}^{\text{master}}$)
endif
```

Device E_k:

```
append($id_i, \mathcal{ID}_k$)

(1) Share protocol keys
$k_{ik}^{\text{attest}} \leftarrow$ hash("attest:"|$k_{ik}^{\text{master}}$)
$k_{ik}^{\text{beat}} \leftarrow$ hash("beat:"|$k_{ik}^{\text{master}}$)

(2a) Find and verify a token in PonA
$id_j \leftarrow$ findNeighbor($\Pi_i$)
if vermac($k_{jk}^{\text{beat}}; \{id_i, q_i, T_A\}, \hat{\mu}_{jk/i}$) then
    if $q_i = q_k$ then
        append($id_i, \mathcal{S}_k$)
        append($id_i, \mathcal{P}_k$)
    endif
endif

(2b) Or verify PoSE
if versig($pk_O; \{id_i, T_{\text{init}}\}, \sigma_i$) then
    if $T_{hb} + \delta_t + t_{tr} - T_{\text{init}} < t_{hb}$ then
        append($id_i, \mathcal{S}_k$)
    endif
endif

(3) Attest new neighbor (if needed)
if $T_A < t_A^{max}$ then
    append($id_i, \mathcal{B}_k$)
else if attest($E_i$)then
    append($id_i, \mathcal{B}_k$)
endif

(4) Create auth and enc keys
if $id_i \in \mathcal{S}_k$ and $id_i \in \mathcal{B}_k$ then
    $k_{ik}^{\text{auth}} \leftarrow$ hash("authenticate:"|$k_{ik}^{\text{master}}$)
    $k_{ik}^{\text{enc}} \leftarrow$ hash("encrypt:"|$k_{ik}^{\text{master}}$)
endif
```

Figure 2: Protocol connect: executed when a device joins or changes its position.

- A signing key-pair (sk_i, pk_i), public key certificate $cert_O(pk_i)$, and public key pk_O of O – sk_i is hardware-protected and is only accessible to US-AID code,
- A reference software configuration c_i indicating correct software that should be present on E_i, and a software configuration certificate $cert_O(c_i)$,
- A Proof-of-Secure-Enrollment (PoSE) π_i, which proves that E_i is not compromised directly after initialization.

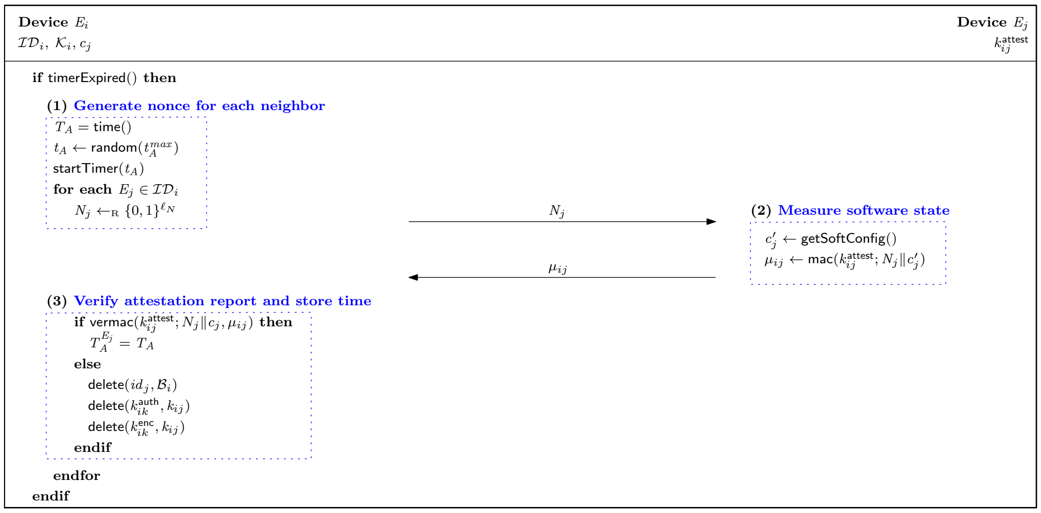
Roaming. As shown in Figure 2, connect has four main modules executed between E_i and E_k: (1) sharing protocol keys, (2a) verifying a token in PonA or (2b) a PoSE, (3) attesting new neighbors, and (4) creating authentication and encryption keys.

In detail, upon joining, or moving within, $\mathcal{E}$, each E_i uses a Key Exchange Protocol KEP (authenticated Diffie-Hellman based on signing key-pairs) to compute, with every neighbor E_k, two protocol keys:

- Attestation key k_{ik}^{attest} used for mutual attestation between E_i and E_k,
- Heartbeat key k_{ik}^{beat} used for authenticating heartbeats and PonA-s exchanged between E_i and E_k.

Two devices also:

- Exchange their respective reference software configurations c_i and c_k as well as certificates: $cert_O(c_i)$ and $cert_O(c_k)$. This allows devices to attest each other at any later point in time.
- Add each other's ID id_k and id_i to their respective list of neighbors $\mathcal{ID}_i$ and $\mathcal{ID}_k$.



```
Device $E_i$                                                        Device $E_j$
$\mathcal{ID}_i,\ \mathcal{K}_i,\ c_j$                              $k_{ij}^{\mathsf{attest}}$

if timerExpired() then
  (1) Generate nonce for each neighbor
    $T_A = \mathsf{time}()$
    $t_A \leftarrow \mathsf{random}(t_A^{max})$
    startTimer($t_A$)
    for each $E_j \in \mathcal{ID}_i$
      $N_j \leftarrow_{\mathrm{R}} \{0,1\}^{\ell_N}$
                         $\xrightarrow{N_j}$                       (2) Measure software state
                                                                     $c'_j \leftarrow \mathsf{getSoftConfig}()$
                                                                     $\mu_{ij} \leftarrow \mathsf{mac}(k_{ij}^{\mathsf{attest}}; N_j \| c'_j)$
                         $\xleftarrow{\mu_{ij}}$
  (3) Verify attestation report and store time
      if vermac($k_{ij}^{\mathsf{attest}}; N_j \| c_j, \mu_{ij}$) then
        $T_A^{E_j} = T_A$
      else
        delete($id_j, \mathcal{B}_i$)
        delete($k_{ik}^{\mathsf{auth}}, k_{ij}$)
        delete($k_{ik}^{\mathsf{enc}}, k_{ij}$)
      endif
    endfor
endif
```

Figure 3: Protocol attest: executed at random times between neighboring devices.

- Exchange PoSE (for newly added devices) or PonA (for roaming devices). Recall that, PoSE-s and PonA-s allow a device to prove to its new neighbors that it has not been compromised.

When E_k receives a PoSE or a PonA from a neighbor E_i during connect, it does the following:

- Verifies its authenticator (MAC or digital signature).
- Verifies its freshness by checking T_{init} (or q_i).
- Decides whether to accept the latest attestation of E_i, or to attest it again, based on T_{init} (or $T_A^{E_i}$).

If authenticator verification and attestation are successful on both devices participating in connect, E_i and E_k then:

- Share two new keys: authentication key k_{ik}^{auth}, and encryption key k_{ik}^{enc}. The list of all keys shared between E_i and its neighbors is denoted by $\mathcal{K}_i$.
- Mark each other as trustworthy: add each other's ID to their $\mathcal{B}$ and $\mathcal{S}$ lists.

Consequently, connect allows benign devices to share authentication and encryption keys with benign neighbors.

Attestation. As shown in Figure 3, attest has three modules executed between E_i and E_j. These are: (1) generating a nonce, (2) measuring software state, and (3) verifying attestation report. In detail, every E_i attests each neighbor E_j at random times (upper bounded by t_A^{max}). attest is executed as follows:

- At attestation time T_A, E_i sends E_j a random challenge N_j.
- E_j replies with a MAC μ_{ij} of its current software configuration c'_j and received challenge.
- If verification of μ_{ij} fails, E_i deletes id_j from its list of benign neighbors $\mathcal{B}_i$. E_i also deletes functionality keys (k_{ij}^{enc} and k_{ij}^{auth}) from set k_{ij} of keys shared with E_j.
- If attestation succeeds, E_i stores the time T_A (as $T_A^{E_j}$) for inclusion in any future PonA sent to E_j.

attest allows each device to continuously check software integrity of all its neighbors and update $\mathcal{B}$ of benign neighbors.

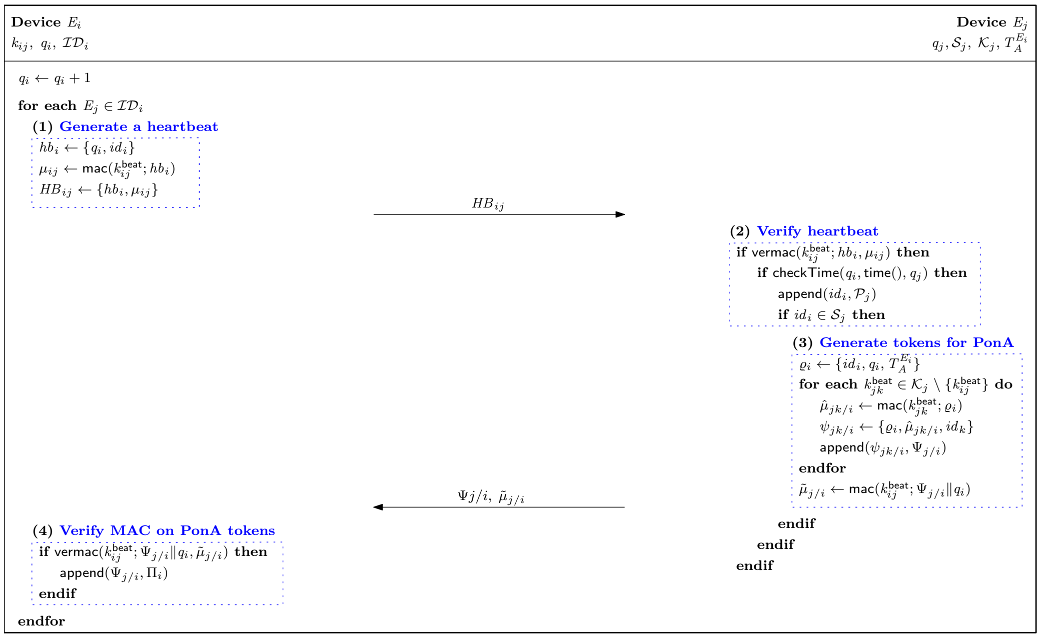
Algorithm 1: checkTime **on** E_j

```
Function checkTime($q_i, t, q_j$)
    if $q_i = q_j \wedge T_{hb} < t < T_{hb} + \delta_t + t_{tr}$ then
        return $true$
    else if $q_i = q_j + 1 \wedge T_{hb} - \delta_t < t < T_{hb}$ then
        return $true$
    else
        return $false$
```

Heartbeat. As shown in Figure 4, beat has four main modules executed between E_i and E_j. These are: (1) generating a heartbeat, (2) verifying a heartbeat, (3) generating tokens for PonA, and (4) verifying MAC on PonA tokens. In detail, each E_i periodically (i.e., at T_{hb}) sends a heartbeat HB_{ij} to every neighbor E_j to prove its presence in $\mathcal{E}$. Upon receiving HB_{ij}, E_j verifies its authenticity, and freshness according to checkTime, as shown in Algorithm 1). This allows E_j to verify that HB_{ij} belongs to current heartbeat interval q_j and is received within accepted tolerance interval t_{tol} around the current T_{hb}. t_{tol} is required to tolerate clock drifts and transmission delays between devices.

If all checks are successful, E_j adds id_i to its list of present devices $\mathcal{P}_j$. It then creates a tuple $\varrho_i = \{id_i, q_i, T_A^{E_i}\}$, and generates MACs over this tuple based on every symmetric key k_{jk}^{beat} shared with each neighbor E_k, thus creating tokens $\psi_{jk/i}$. The set of all created tokens is then sent to E_i, which stores them along with other tokens received from all other neighbors into a PonA Π_i. At the end of t_{tol}, each E_j compares IDs of its neighbors that have not been physically attacked $\mathcal{S}_j$ to the list of present neighbors $\mathcal{P}_j$. Every id_i not present in $\mathcal{P}_j$ is deleted from $\mathcal{S}_j$, and devices' corresponding functionality keys (k_{ij}^{auth} and k_{ij}^{enc}) are deleted from k_{ij}.

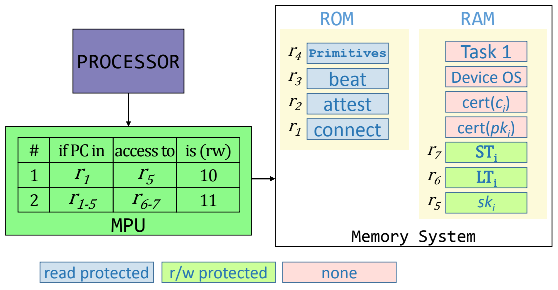
The beat protocol allows every device to periodically check presence of each neighbor and its hardware trustworthiness, and then update $\mathcal{S}$. Also, Π_i acquired by every E_i after beat allows it to later prove its hardware and software trustworthi-



```
Device $E_i$                                                    Device $E_j$
$k_{ij},\ q_i,\ \mathcal{ID}_i$                                 $q_j, \mathcal{S}_j,\ \mathcal{K}_j,\ T_A^{E_i}$

$q_i \leftarrow q_i + 1$
for each $E_j \in \mathcal{ID}_i$
  (1) Generate a heartbeat
  $hb_i \leftarrow \{q_i, id_i\}$
  $\mu_{ij} \leftarrow \mathsf{mac}(k_{ij}^{\mathsf{beat}}; hb_i)$
  $HB_{ij} \leftarrow \{hb_i, \mu_{ij}\}$
                          $\xrightarrow{HB_{ij}}$
                                                  (2) Verify heartbeat
                                                  if $\mathsf{vermac}(k_{ij}^{\mathsf{beat}}; hb_i, \mu_{ij})$ then
                                                      if $\mathsf{checkTime}(q_i, \mathsf{time}(), q_j)$ then
                                                          $\mathsf{append}(id_i, \mathcal{P}_j)$
                                                          if $id_i \in \mathcal{S}_j$ then
                                                              (3) Generate tokens for PonA
                                                              $\varrho_i \leftarrow \{id_i, q_i, T_A^{E_i}\}$
                                                              for each $k_{jk}^{\mathsf{beat}} \in \mathcal{K}_j \setminus \{k_{ij}^{\mathsf{beat}}\}$ do
                                                                  $\hat{\mu}_{jk/i} \leftarrow \mathsf{mac}(k_{jk}^{\mathsf{beat}}; \varrho_i)$
                                                                  $\psi_{jk/i} \leftarrow \{\varrho_i, \hat{\mu}_{jk/i}, id_k\}$
                                                                  $\mathsf{append}(\psi_{jk/i}, \Psi_{j/i})$
                                                              endfor
                                                              $\tilde{\mu}_{j/i} \leftarrow \mathsf{mac}(k_{ij}^{\mathsf{beat}}; \Psi_{j/i} \| q_i)$
                          $\xleftarrow{\Psi j/i,\ \tilde{\mu}_{j/i}}$
  (4) Verify MAC on PonA tokens
  if $\mathsf{vermac}(k_{ij}^{\mathsf{beat}}; \Psi_{j/i} \| q_i, \tilde{\mu}_{j/i})$ then
      $\mathsf{append}(\Psi_{j/i}, \Pi_i)$
  endif
                                                          endif
                                                      endif
                                                  endif
endfor
```

Figure 4: Protocol beat: executed periodically between neighboring devices.



#	if PC in	access to	is (rw)
1	r_1	r_5	10
2	$r_{1\text{-}5}$	$r_{6\text{-}7}$	11

Figure 5: Implementation of US-AID on SMART [10]

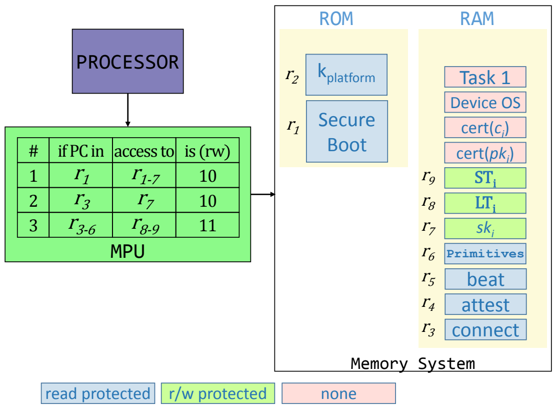
ness to any new neighbor via connect.

IV. Implementation

We implemented US-AID on two security architectures for low-end embedded devices: SMART [10] and TrustLite [16] in order to confirm its viability and evaluate performance. However, since SMART and TrustLite are not available on commodity devices, we also implemented US-AID on a small autonomous network formed of six Raspberry Pi-based drones communicating over WiFi. This implementation allowed us to further demonstrate practicality of US-AID. Finally, in order to evaluate US-AID for very large networks we used network simulations based on measurements from our SMART and TrustLite implementations. In the rest of this section we describe our implementations.

A. SMART & TrustLite Overview

SMART [10] (see Figure 5) is an architecture for secure remote attestation on low-end embedded systems based on minimal hardware assumptions. Main hardware components of SMART are: (1) read-only memory (ROM), which stores program code used for attestation (denoted by ROM code) and



#	if PC in	access to	is (rw)
1	r_1	$r_{1\text{-}7}$	10
2	r_3	r_7	10
3	$r_{3\text{-}6}$	$r_{8\text{-}9}$	11

Figure 6: Implementation of US-AID on TrustLite [16]

an attestation key. ROM provides immutability and ensures integrity of the attestation code; and (2) simple memory protection unit (MPU), which controls access to the attestation key. MPU grants access to the key only to ROM code based on the value of the program counter. Consequently, only attestation code can access the attestation key. SMART also provides additional features for protecting the attestation key, e.g., it ensures that ROM code is executes in an uninterrupted and atomic fashion, starting from the first, and terminating in its last, instruction.

TrustLite [16] (see Figure 6) is an embedded security architecture which generalizes SMART allowing isolated execution of multiple tasks (processes). Isolation of tasks is attained via *secure boot* and an execution-aware memory protection unit (EA-MPU), which is set up by access rules between tasks and their corresponding data. Changes to EA-MPU are prevented by starting the system via secure boot, which verifies that correct software sets up the rules in EA-MPU and sets EA-MPU configurations registers to read-only to prevent further changes. The main advantage of TrustLite over SMART is that the former allows secure interruption of tasks while executing.

B. Implementation Details

SMART-specific Extensions. To implementat US-AID, we needed to extend SMART to enable restricted access to writable memory, and multiple entry and exit points of trusted ROM code. The former is needed to store symmetric keys shared with neighboring devices, in addition to US-AID's intermediate data. It was done by extending the MPU to control access to a small area of writable non-volatile memory. The latter is needed to run different protocols independently, from within the same ROM. It was done by extending hardware entry and exit points with software entry points. We implemented the protocol as a single function called US-AID, which takes as input the identity of the protocol, and always exits from the same address. Note that no such extensions were required for TrustLite, since it already provides secure writable memory and multiple entry and exit points for trusted code.

Keys and Variables. The secret key sk_i, is both read- and write-protected by a dedicated EA-MPU rule (rule #1 in Figure 5 and rule #2 in Figure 6), which ensures that connect has exclusive read access to sk_i while all other code has neither read nor write access to sk_i. Long term $\mathbf{LT}_i$ and short term $\mathbf{ST}_i$ protocol data (including symmetric keys and variables) are also both read- and write-protected through dedicated EA-MPU rules, (see Figure 5 and 6). At the same time, protocol code integrity is assured by ROM (on SMART [10]) and secure boot (on TrustLite [16]).

Real-Time Clock. As mentioned earlier, secure in-network attestation and absence detection require a real-time clock. The clock must be write-protected and must not wrap around within the lifetime of a device. For example, a 64-bit register would wrap around in $12,186.3$ years on our 48 MHz TrustLite if incremented every clock cycle. On the other hand, a 32-bit register can be made to wrap around every 3 years, if incremented every one million cycle, i.e., providing a resolution of 21 ms, which is appropriate for US-AID.

Functionality. In order to reflect the outcome of attestation and absence detection in the network functionality and provide real isolation of detected malicious device, we implemented an additional component – **`Primitives`**. It provides authentication (authenticate, and verify in Algorithm 2 and 3) and encryption services to all software components residing on a device and contributing to its functionality, without granting them direct access to shared symmetric keys. In particular, before performing the required action (i.e., authenticate or encrypt), **`Primitives`** on E_i checks whether the ID id_j of E_j, to which E_i is communicating, exists in both $\mathcal{B}_i$ and $\mathcal{S}_i$.

Implementation on drones testbed. To further test and demonstrate feasibility of US-AID, we implemented and evaluated US-AID on an autonomous testbed composed of six

Algorithm 2: authenticate **on** E_i

```
1 Function authenticate ($m, id_j$)
2     if $id_j \in \mathcal{S}_i$ and $id_j \in \mathcal{B}_i$ then
3         return mac($k_{ij}^{\text{auth}}; m$)
```

Algorithm 3: verify **on** E_i

```
1 Function verify ($m, \mu, id_j$)
2     if $id_j \in \mathcal{S}_i$ and $id_j \in \mathcal{B}_i$ then
3         return vermac($k_{ij}^{\text{auth}}; m, \mu$)
```

drones that form an ad-hoc network as shown in Figure 7. Each drone is equipped with a Raspberry Pi 3 Model B with a 1.2 GHz Quad-core 64-bit CPU. Furthermore, drones communicate via WiFi using a 150 MBit/s USB WLAN stick. Each drone is initialized with an RSA private key, a public key certificate, and the public key of a network operator O. We used C programming language and utilized mbed TLS [3] library for handling cryptographic operations.

V. Performance Evaluation

We evaluated US-AID based on implementations described in Section IV. We now present evaluation results for TrustLite. Since results for SMART are very similar, they are omitted, due to space limitations.

Software Complexity and Memory Requirements. All cryptographic operations in US-AID (i.e., authentication and encryption) already exist on TrustLite. We only need to add some code responsible for creating, sending, and handling beat, attest and connect messages, as well as handling lists $\mathcal{K}$, $\mathcal{ID}$, $\mathcal{P}$, $\mathcal{S}$, and $\mathcal{B}$. Letting g denote the number of neighbors of each device, each device need to store $36 \cdot g^2 + 56 \cdot g + 228$ bytes of long-term **LT** and short-term **ST** protocol data, $56 \cdot g + 80$ bytes of which should be read- and/or write-protected.

Hardware Costs. We compare our implementation to the current implementation of TrustLite [16]. Results are shown in Table I.

Table I: Hardware cost of US-AID

	TrustLite	US-AID	% of increase
Register	6038	6186	2.45%
Look-up Table	15142	15356	1.41%

The total cost of US-AID is $6,186$ registers and $15,356$ LUTs, which is a very small increase of 2.45% and 1.41%, respectively, over the cost of TrustLite in terms of registers and LUTs respectively.

Energy Costs. Energy consumption estimates[4] of US-AID are shown in Figure 8. We base it on previously reported

[4]It is not possible to analyze the energy consumption of US-AID directly on SMART [10] and TrustLite [16] since both are only available on FPGAs.

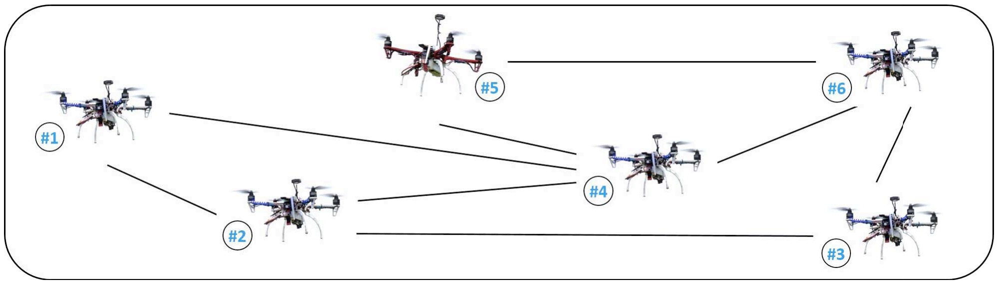


Figure 7: Our autonomous drones testbed

energy consumption for communication and cryptographic operations of TelosB and MICAz sensor nodes [9] which fall into the same class of low-end embedded devices as supported by SMART [10] and TrustLite [16]. Measurements exclude energy for generating the software configuration and executing a key exchange protocol, which is dependent on the specific KEP protocol used. Energy consumption of one execution of all of protocols is linear in the number of neighbors, except for move (i.e., connect for a moving device), energy consumption of which is cubic in the number of neighbors. This is mainly due to the size of the Proofs-of-non-Absence (PonA-s). Energy consumption in all protocols is constant in the size of the network and can be as low as 1, 5, and 20 mJ for attest, join (i.e., connect for a new device) / beat, and move, respectively, in networks with up to 4 neighbors per a MICAz device.

Figure 9 shows the energy consumption of beat (executed with 8 neighbors) as a function of the number of heartbeat intervals elapsing within a specific period of time, where a heartbeat interval is the time between two consecutive executions of beat protocol. The energy consumption of beat grows linearly with the number of heartbeat intervals. The length of a heartbeat interval depends on anticipated adversary's budget and devices' hardware complexity. In other words, the length of a heartbeat interval represents a trade-off between security and performance. A short heartbeat interval provides more accurate detection of physical attacks by adversaries with bigger budgets. However, it also implies additional energy consumption which grows linearly with the number of these intervals.

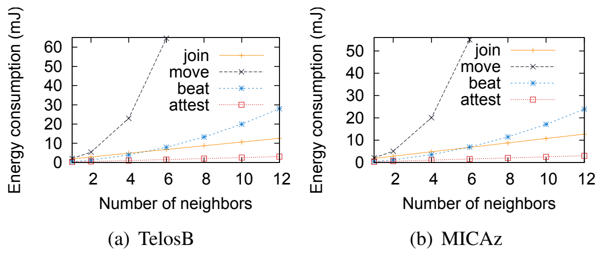


Figure 8: Performance evaluation of US-AID

Run-Time of Primitives. Table II presents an evaluation of **`Primitives`** on TrustLite. Time to authenticate and encrypt a 64Byte message can be as low as 780 μs. On the other hand, verifying and decrypting such a message requires only $1,190$ μs.

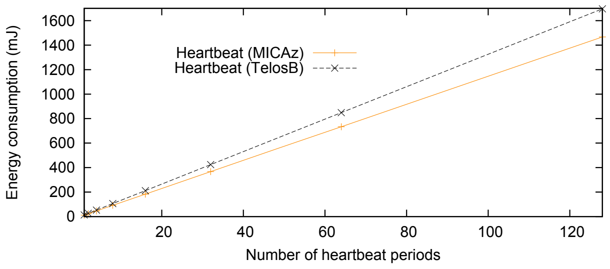


Figure 9: Energy consumption of beat

Table II: Run-time of **`Primitives`**

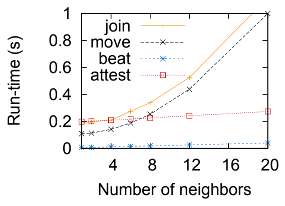
Run-time at 48 MHz TrustLite [16] (μs) for 64 Byte messages			
authenticate	verify	encrypt	decrypt
320	320	460	870

Simulation Results. We used OMNeT++ [19] simulation environment to evaluate performance of US-AID. Cryptographic operations were simulated as delays based on real measurements from SMART [10] and TrustLite [16] implementations. Networks with up to $1,000,000$ devices and variable number of neighbors were simulated. We assume that each device has 100 KB of memory to be attested. We exclude the time to execute a key exchange protocol KEP since it does not present additional overhead[5] and it depends on the specific KEP used. Results are shown in Figure 10 and 11.

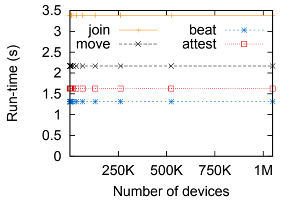
Run-times of beat and attest are linear, while run-times of join and move are quadratic in the number of neighbors. This is due to the exchange of local PonA-s, the size of which is quadratic in the number of neighbors (see Figure 10(a)). Figure 10(b) shows the run-time of US-AID in networks with 12 neighbors per device. The run-time of all US-AID protocols is independent of the network size, which makes it scalable.

Finally, Figure 11 shows the run-time of US-AID in comparison to the closest related work — SEDA [5] and DARPA [13]. While run-times of attest and beat in US-AID are low and constant in network size, run-time of attest (described in SEDA) and the most efficient version of collect (described in

[5]Regardless of US-AID, devices must share keys. This overhead is also shared with all existing collective attestation schemes [5], [1], [13], [6].

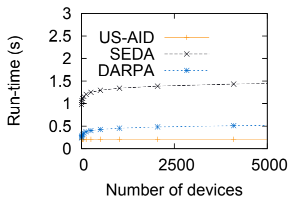


(a) Per neighbors

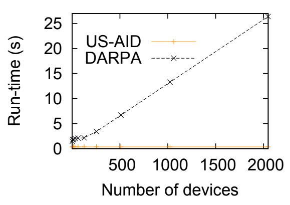


(b) Per network size

Figure 10: Run-time of US-AID



(a) attest



(b) beat

Figure 11: US-AID vs. SEDA and DARPA

DARPA) is logarithmic in the network size. Note that, collect has a linear verification time for the verifier (Figure 11(a)). On the other hand, the most efficient version of heartbeat (described in DARPA) has a linear run-time, as compared to constant run-time of beat in US-AID (Figure 11(b)).

Run-time on Autonomous Drones Network. US-AID run-times in our testbed are much faster than those of SMART and TrustLite (targeted by US-AID) due to the relatively powerful CPU of our Raspberry Pis. For the sake of completeness, we list run-times for all protocols.[6] The run-time of beat of drone #5 is less than 4 ms; the run-time of attest of drone #6 to attest (100 KB of code on) drone #3, #4 and #5 is less than 17 ms; and the run-time of move (i.e., connect for a moving device) of drone #3 to connect to drone #1 is less than 12 ms (see Figure 7).

VI. Security Consideration

An adversary can break the security goal of US-AID (i.e., isolation) and convince a benign E_b to securely communicate with a malicious E_m, only if: (1) E_m is one of E_b's existing neighbors which sends correct heartbeats at correct times, and is successfully attested; or (2) E_m is a mobile or a newly added device, which proves trustworthiness of its state through a correct PoSE or PonA. We consider the following cases:

$\mathcal{A}$ Attacks beat. Non-negligible tampering time implies that, in order to physically attack E_m, $\mathcal{A}$ must detach it for a non-negligible amount of time, which is greater than the accepted tolerance interval t_{tol} around T_{hb}; see Section III for details. Therefore, $\mathcal{A}$ should fake or replay a correct heartbeat HB_{mb} in order to convince a current neighbor E_b that E_m is not absent. This would require $\mathcal{A}$ to either forge the MAC μ_{mb}, or find and replay an old MAC that is valid for a fresh heartbeat interval q_m, which is periodically incremented.

[6]Results are an average over 100 executions

$\mathcal{A}$ Attacks attest. The time between two consecutive attestation instances is upper-bounded by $t_{\mathcal{A}}^{max}$. To keep E_m connected to the secure sub-network after its compromise, $\mathcal{A}$ must fake or replay a valid attestation response μ_{mb} over a benign software configuration c'_m of E_m. This would require $\mathcal{A}$ to either forge the MAC μ_{mb}, or find and replay an old MAC that is still valid over a fresh random nonce and a benign software configuration, or find a collision for the hash function used to measure software state and generate c'_m.

$\mathcal{A}$ Attacks move (i.e., connect for a Moving Device). E_m and E_b can only become neighbors if they can prove to each other, through a token ($\psi_{jb/m}$ for E_m and $\psi_{jm/b}$ for E_b) in their PonA-s, that they have been present in all previous heartbeat intervals. Since each token in PonA should be authenticated with a MAC μ_{jb} by a common neighbor E_j, which is in the secure sub-network, $\mathcal{A}$ must fake or reuse $\psi_{jb/m}$. This would require $\mathcal{A}$ to either forge the MAC μ_{jb}, or find and replay an old MAC that is valid over a fresh heartbeat interval q_m.

$\mathcal{A}$ Attacks join (i.e., connect for a New Device). A newly added E_m only acquires neighbors if: (1) it can prove the trustworthiness of its state through a PoSE π_m; and (2) every new neighbor E_b in its first heartbeat interval can prove its presence through a token $\psi_{jm/b}$ in PonA. Consequently, in order to add E_m to the secure sub-network, $\mathcal{A}$ must fake or replay a π_m. This would require $\mathcal{A}$ to either forge σ_m of O, or find and replay a digital signature that is still valid over a fresh timestamp T_{init}. Similarly, to add E_m as a bridge between the secure sub-network and malicious devices, $\mathcal{A}$ must fake or replay $\psi_{jb/m}$. This would require either forging a MAC μ_{jb}, or replaying a MAC that is valid over a fresh heartbeat interval q_m.

VII. Related Work

Remote Attestation. In a remote attestation protocol, a trusted *verifier* obtains an authenticated report about the current software state of a remote and possibly compromised device *prover*. A trust anchor on the prover is required to guarantee the integrity of the attestation code. Typically the trust anchor is implemented in hardware, e.g., based on Trusted Platform Modules (TPM). However, TPMs and other similar modules are complex and expensive [27], [17] and not suitable for low-end embedded devices. On the other hand, software-based attestation [22], [18] schemes require no hardware security features and are suitable for attesting non-remote embedded devices. Also, these schemes are based on strong assumptions that are hard to achieve in practice [4]. Finally, hybrid attestation methods [10], [11], [16] provide stronger security guaranties while requiring minimal hardware security features. All of the above assume a single prover device.

Collective Attestation. A collective attestation protocol is a remote attestation protocol, with many provers [5], [1], [13], [6]. SEDA [5] was the first step towards collective attestation. It performs attestation of swarms of interconnected embedded devices by distributing attestation burden across the entire network. Unfortunately, SEDA considers only remote software attacks and fails as soon as a single device is physically attacked. SANA [1] combines SEDA with a novel aggregate signature scheme which makes it more resilient to physical attacks. DARPA [13] builds on top of SEDA by extending it with an absence detection protocol, which leads to detection of physical attacks. However, its computation and communication overhead is quadratic in terms of network size. US-AID identifies physical attacks also based on absence detection. However, it enables efficient detection of physical attacks based on local heartbeats. Also, by combining frequent attestation and absence detection with key exchange, US-AID detects all software-compromised and/or physically-attacked devices in autonomous dynamic networks.

Attestation & Key Exchange. There is a lot of prior work that combines attestation with key exchange [22], [20]. SAKE [22] aims to establish shared keys between neighboring nodes in sensor networks without relying on any prior shared secrets. Key exchange in SAKE is based on the attestation result of one involved sensor node. Unfortunately, SAKE relies on a software-based attestation security of which is based on strong and unrealistic assumptions [4]. [20] proposes extending IPsec [15] key exchange protocol IKEv2 [14] to support attestation. The goal of this extension is to ensure the software trustworthiness of end-points while an IPsec connection is running. However, this extension targets legacy networks of high-end computing platforms and is thus not applicable to autonomous dynamic networks of embedded systems.

Absence Detection. Absence detection has been studied in the context of Wireless Sensor Networks (WSNs) where it was used for detecting node failures. There are several prior techniques for both static [26] and dynamic topologies (e.g., [12]). However, these are not designed with security in mind, and are thus ineffective in our adversarial setting. Furthermore, there are WSN-focused techniques that use absence detection to identify captured devices [7], [8]. However, some are probabilistic and allow false negatives in dynamic networks, while others are only suitable for static networks, and cannot be easily extended to dynamic ones. The PonA protocol proposed in this paper provides one such extension. Finally, one common drawback of all prior work is that they not provide security in our stronger adversary model, since they are all vulnerable to remote software attacks.

VIII. Conclusions

Current remote attestation techniques are not scalable to large networks of embedded devices. To this end, several collective attestation techniques have been recently proposed. However, these techniques are not applicable to large, *autonomous dynamic* IoT networks. In this paper, we constructed US-AID – the first collective attestation scheme for such networks. We reported on implementations of US-AID on recent security architectures for low-end embedded devices, showing additional hardware cost. Furthermore, we demonstrated US-AID's feasibility by implementing and testing it on an autonomous ad-hoc network of six inter-connected drones. We also assessed performance of US-AID in terms of energy and run-time, based on real measurements and simulations of up to $1,000,000$-device networks.

Acknowledgments

This research was co-funded by the German Science Foundation, as part of project S2 within CRC 1119 CROSSING, HWSec, and Intel Collaborative Research Institute for Collaborative Autonomous & Resilient Systems (ICRI-CARS). Gene Tsudik was supported in part by: (1) DHS under subcontract from HRL Laboratories, (2) ARO under contract W911NF-16-1-0536, and (3) NSF WiFiUS Program Award 1702911.

References

[1] M. Ambrosin et al. SANA: Secure and Scalable Aggregate Network Attestation. In *ACM CCS*, 2016.
[2] W. Arbaugh et al. A secure and reliable bootstrap architecture. In *IEEE S&P*, 1997.
[3] ARM Limited. Ssl library mbed tls / polarssl. https://tls.mbed.org/.
[4] F. Armknecht et al. A security framework for the analysis and design of software attestation. In *ACM CCS*, 2013.
[5] N. Asokan et al. Seda: Scalable embedded device attestation. In *ACM CCS*, 2015.
[6] X. Carpent et al. Lightweight swarm attestation: A tale of two lisa-s. In *ASIACCS*, 2017.
[7] M. Conti et al. Emergent properties: Detection of the node-capture attack in mobile wireless sensor networks. In *WiSec*, 2008.
[8] M. Conti et al. Mobility and cooperation to thwart node capture attacks in manets. *EURASIP WCN*, 2009.
[9] G. de Meulenaer et al. On the energy cost of communication and cryptography in wireless sensor networks. In *WiMob*, 2008.
[10] K. Eldefrawy et al. SMART: Secure and minimal architecture for (establishing a dynamic) root of trust. In *NDSS*, 2012.
[11] A. Francillon et al. A minimalist approach to remote attestation. In *DATE*, 2014.
[12] N. Hayashibara et al. Failure detectors for large-scale distributed systems. In *SRDS*, 2002.
[13] A. Ibrahim et al. DARPA: Device Attestation Resilient against Physical Attacks. In *WiSec*, 2016.
[14] C. Kaufman. Internet Key Exchange (IKEv2) Protocol. RFC 4306 (Proposed Standard), 2005.
[15] S. Kent et al. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), 2005.
[16] P. Koeberl et al. TrustLite: A security architecture for tiny embedded devices. In *EuroSys*, 2014.
[17] X. Kovah et al. New results for timing-based attestation. In *IEEE S&P*, 2012.
[18] Y. Li et al. VIPER: Verifying the integrity of peripherals' firmware. In *ACM CCS*, 2011.
[19] OpenSim Ltd. OMNeT++ discrete event simulator. http://omnetpp.org/.
[20] A.-R. Sadeghi et al. Extending ipsec for efficient remote attestation. In *FC*, 2010.
[21] D. Schneider. Jeep Hacking 101. http://spectrum.ieee.org/cars-that-think/transportation/systems/jeep-hacking-101, 2015.
[22] A. Seshadri et al. SAKE: Software attestation for key establishment in sensor networks. In *IEEE DCOSS*, 2008.
[23] S. Skorobogatov. Physical attacks on tamper resistance: Progress and lessons. In *Workshop on Hardware Assurance*, 2011.
[24] S. Skorobogatov. Physical attacks and tamper resistance. In *Introduction to Hardware Security and Trust*. Springer, 2012.
[25] S. P. Skorobogatov. *Semi-invasive attacks: a new approach to hardware security analysis.* PhD thesis, University of Cambridge, 2005.
[26] P. Stelling et al. A fault detection service for wide area distributed computations. *Cluster Computing*, 1999.
[27] Trusted Computing Group (TCG). Website. http://www.trustedcomputinggroup.org, 2015.
[28] J. Vijayan. Stuxnet renews power grid security concerns, 2010.
[29] Y. Zhou et al. Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing. *IACR Cryptology ePrint Archive*, 2005.