Establishing Mutually Trusted Channels for Remote Sensing Devices with Trusted Execution Environments

Carlton Shepherd, Raja Naeem Akram, Konstantinos Markantonakis
{carlton.shepherd.2014,r.n.akram,k.markantonakis}@rhul.ac.uk
Information Security Group, Royal Holloway, University of London
Egham, Surrey, United Kingdom

ABSTRACT

Remote and largely unattended sensing devices are being deployed rapidly in sensitive environments, such as healthcare, in the home, and on corporate premises. A major challenge, however, is trusting data from such devices to inform critical decision-making using standardised trust mechanisms. Previous attempts have focused heavily on Trusted Platform Modules (TPMs) as a root of trust, but these forgo desirable features of recent developments, namely Trusted Execution Environments (TEEs), such as Intel SGX and the GlobalPlatform TEE. In this paper, we contrast the application of TEEs in trusted sensing devices with TPMs, and raise the challenge of secure TEE-to-TEE communication between remote devices with mutual trust assurances. To this end, we present a novel secure and trusted channel protocol that performs mutual remote attestation in a single run for small-scale devices with TEEs. This is evaluated on two ARM development boards hosting GlobalPlatform-compliant TEEs, yielding approximately four-times overhead versus untrusted world TLS and SSH. Our work provides strong resilience to integrity and confidentiality attacks from untrusted world adversaries, facilitates TEE interoperability, and is subjected to mechanical formal analysis using Scyther.

CCS CONCEPTS

•**Security and privacy** → **Trusted computing;** *Embedded systems security; Domain-specific security and privacy architectures;* •**Networks** → Security protocols;

KEYWORDS

Trusted sensing, trusted execution environments, trusted computing, embedded device security, secure sensing architectures.

ACM Reference format:
Carlton Shepherd, Raja Naeem Akram, Konstantinos Markantonakis. 2017. Establishing Mutually Trusted Channels for Remote Sensing Devices with Trusted Execution Environments. In *Proceedings of ARES '17, Reggio Calabria, Italy, Aug. 29–Sep. 01, 2017,* 10 pages.
DOI: 10.1145/3098954.3098971

ARES '17, Reggio Calabria, Italy

978-1-4503-5257-4/17/08…$15.00
DOI: 10.1145/3098954.3098971

1 INTRODUCTION

The advent of the Internet of Things (IoT) – the notion that interconnected everyday objects will monitor and act upon their surroundings – is enabling pervasive sensing applications to become reality. Healthcare proposals have long existed that transmit physiological data from wearable sensors for self-monitoring [46], early detection of illness [32], and to inform insurance premiums [8]. In the home, energy conservation systems using occupancy data [22], task automation for the elderly and disabled [34], and fall/injury detection [31] have been proposed. This extends to monitoring transported goods, e.g. perishable foods [5] and livestock [29], smart manufacturing [24], and various other domains. Across sensor-driven domains, however, the challenge of data assurance is presented: whether remote sensing devices can be trusted to inform critical decision-making in which malicious data could yield damaging consequences for end-users.

This has motivated past work using Trusted Computing Group (TCG) concepts, namely the Trusted Platform Module (TPM), to protect data and provide evidence of platform integrity. TPMs, however, are restricted relative to modern trust technologies: neither arbitrary application execution nor secure input/output (I/O) are realisable without additional processes, such as TPM-backed virtual machines [48]. One solution is the Trusted Execution Environment (TEE), which shares hardware with an untrusted Rich Execution Environment (REE), e.g. Android, but executes independently with hardware-enforced isolation. This allows critical applications to execute with strong confidentiality and integrity guarantees alongside a potentially compromised REE, while providing TPM-like features, such as secure storage and authenticated boot.

In this paper, we investigate TEE-based sensing to achieve certain trust and security assurances, contrast these with TPMs (Sections 2 and 3), and identify the challenges facing their use. Particularly, the issue of mutual trusted channels for TEEs – the notion of secure TEE-to-TEE communication with trust assurances – has received little attention in past literature. To address this, we present a novel protocol that provides secure and trusted channels between two remote TEEs (Sections 4 and 5), where trust can be verified *unidirectionally* or *bi-directionally* through one- and two-way remote attestation respectively. This provides resilience to sophisticated integrity and confidentiality software attacks from intermediate components, i.e. REE or network-level adversary. Our work is applicable to sensing applications requiring stronger trust assurances, and we discuss measures for facilitating interoperability between competing TEEs, such as Intel SGX and the GlobalPlatform TEE. The protocols are formalised and subjected to mechanical formal analysis using Scyther (Section 5), before presenting a performance evaluation using a GlobalPlatform-compliant TEE on two ARM

development boards (Sections 6 and 7). Lastly, we conclude our findings and identify future research directions (Section 8).

2 RELATED WORK

Sensing with standardised trust technologies has attracted notable attention in past literature; we discuss relevant work and highlight their contributions. Liu et al. [26] propose two abstractions for providing trust in sensing platforms: **(1)**, *sensor attestation*, which signs each reading with a TPM's Attestation Identity Key (AIK) and verified with its public key; platform integrity is preserved using regular remote attestation (described in Section 3). **(2)**, *Sensor seal*: encrypted secrets are bound to the TPM and released once sensor readings satisfy a given policy. These two concepts are evaluated using ARM TrustZone with a TPM emulated in software, and Credo, a TPM-backed hypervisor, for X86 systems. Both are evaluated yielding moderate overhead.

Gilbert et al. [14] examine TPMs to provide data assurance in sensitive sensing applications. Similarly, the authors propose a TPM-backed hypervisor to protect device drivers and sensing processing applications from software integrity attacks. Untrusted user applications are executed in a separate guest domain to facilitate isolated execution, with the hypervisor acting as a secure monitor. It is proposed to sign sensing measurements with a signed TPM quote, which includes the TPM Platform Configuration Register (PCR) responsible for the sensing application.

Saroiu et al. [40] propose two sensing architectures using TPMs: **(1)**, a TPM-backed hypervisor that sandboxes users' software in a guest Virtual Machine (VM), and a root VM with privileged access to sensor drivers and the TPM. The root VM signs sensor readings with the TPM to prevent modification once passed to the user VM. Design **(2)** proposes attaching a TPM to each sensing microcontroller to protect against a malicious kernel. Here, the microcontroller signs readings using the TPM independently of the OS. The authors note that **(1)** is less costly – both performance-wise and economically – but provides fewer security guarantees.

To our knowledge, the use of TEEs in trusted sensing has avoided direct examination in light of recent developments, such as Intel SGX and the GlobalPlatform TEE. Certain TEE features have already been identified as desirable, e.g. secure I/O and isolated execution [14, 26]. In other work, these features have been developed independently with TPMs [21, 26, 28, 48], but a unified TPM-based system has not materialised as a deployable solution over time. The shortfall of TPMs for such general-purpose trusted computing is attributed to the large and ever-changing Trusted Computing Base (TCB) necessary to maintain it [38]. A large TCB – the software and hardware components critical to a system's security – increases the attack surface available to an adversary. For sensing applications, a compromise at any point would undermine data assurance, i.e. whether it was modified or breached confidentiality.

3 TRUSTED SENSING PLATFORMS

The lack of trust assurances in sensing platforms is a major challenge in deploying services that could significantly benefit users [40]. One example is enabling healthcare clinicians to suggest treatments via users' physiological data reported from health wearables [39]. Another is the use of sensor-driven authentication schemes to control access to remote assets based on location, application activity or other behavioural data [42, 44]. In such examples, protecting platform integrity is paramount to prevent data manipulation at source, as well as protecting measurements at rest or in transit, e.g. over Bluetooth or WiFi.

Table 1: Comparison of TPM and TEE Features.

Feature	Rich OS	TPM	SGX	GP TEE +TrustZone
HW Tamper-Resistance	✗	✓	◇	◇
Native Isolated App. Execution	✗	✗	✓	✓
No Additional Hardware	✓	✗	✓	✓
Remote Attestation	✗	✓	✓	✗
Native Secure I/O	✗	✗	✗	✓
Native Secure Storage	✗	◇	✓	✓
Authenticated Boot	✗	✓	✗	✓

◇ Limited functionality.

To complicate matters, reported measurements are often several steps removed from raw values. Feature extraction and preprocessing algorithms, such as noise filtering and compression, are applied regularly before transmission [14]. This places TPMs at a disadvantage, which do not naturally provide isolated execution to transform sensitive data securely. Moreover, data may be collected off-line and stored persistently before transmission. This is beneficial where real-time streaming is prohibitive due to battery and network constraints, especially for detailed, high-volume measurement streams. In such cases, measurements ought to be stored securely (on-device) to preserve integrity and confidentiality. This too is problematic with TPMs, however, which provide limited on-board storage and processing capability for encrypting potentially feature-rich measurements at high sampling rates. An approach based on encrypting measurements, as in [26], necessitates a trusted path with sensor hardware; a compromised I/O driver, for example, may intercept data before it reaches a TPM.

TEEs offer promise by providing standardised secure I/O, secure storage and isolated execution. This allows sensing nodes to robustly store and process sensitive material while preserving performance through hardware sharing. TEEs may also offer other TPM-like mechanisms, such as authenticated boot and remote attestation. We provide a comparison of TPMs and TEEs in Table 1. Next, we explore TEEs further and their challenges with respect to trusted sensing.

3.1 Trusted Execution Environments (TEEs)

GlobalPlatform defines a TEE as an execution environment, isolated from the REE, which *"protects from general software attacks, defines rigid safeguards as to data and functions that a program can access, and resists a set of defined threats"* [15]. TEEs are used to execute sensitive applications, such as fingerprint matching, with hardware-enforced isolation via a trusted hardware element, e.g. a CPU. Trusted applications (TAs) reside in their own memory regions and such isolation is used to prevent unauthorised accesses from REE space; TAs may allocate shared memory space with trusted world applications, or expose developer-defined functions that are mediated by a high-privilege secure monitor. TEEs are finding development on devices suitable for sensing platforms.

Micro-controllers, such as the Atmel SMART[1], and single-board computers, including the Raspberry Pi 3[2], contain ARM TrustZone extensions. Intel SGX, meanwhile, is finding adoption on server-side and larger portable devices, e.g. Intel-based laptops. A detailed comparison of secure and trusted environments is found in [43]. We summarise these main TEEs below.

3.1.1 Intel Software Guard eXtensions (SGX). An extension to the x86-64 instruction set that allows the creation of independent TEEs, or 'enclaves', per application on-demand. Enclaves reside in isolated memory regions within RAM with accesses mediated by a (trusted) CPU. Enclaves may access untrusted world memory space directly, but not vice-versa; enclaves may also not access other enclaves arbitrarily. Intel SGX offers secure storage using the 'sealing' abstraction in which enclave data is encrypted using a key derived from the CPU-bound root seal key, where 'sealed' data is accessible only to that enclave. SGX offers remote attestation for third-party verification of enclave integrity using the Enhanced Privacy ID (EPID) protocol [7] – a variant of Direct Anonymous Attestation (DAA) that uses a group signature scheme with a CPU-bound EPID key for preserving platform privacy. EPID bootstraps a key for allowing remote operators to provision or retrieve sensitive data from the target enclave. SGX is available from the Skylake generation of Intel CPUs [9].

3.1.2 GlobalPlatform (GP) TEE and ARM TrustZone. Maintains two worlds for all trusted and untrusted applications. The trusted world contains a trusted OS that interfaces with TAs through the GP TEE Internal API (see Figure 1 and [18]), while untrusted world applications interface with the trusted world using the Client API [15]. The GP TEE provides secure storage using the sealing abstraction, akin to SGX; a trusted user-interface API for establishing secure paths between TAs and a touch display; a Sockets API for initiating TEE network connections using POSIX-style sockets; and a Secure Element API for communicating with on-board secure elements [19]. ARM TrustZone is typically used to instantiate GP TEEs in hardware, which uses two virtual cores for each world per physical CPU core and an additional Non-Secure (NS) processor bit to indicate execution mode. Secure Monitor Call (SMC) instructions are used to context switch between worlds. Moreover, TrustZone uses secure ROM for performing authenticated boot of the trusted world by measuring and computing a signature chain of critical system components, e.g. a bootloader. Secure I/O is provided through which dedicated interrupts can be routed specifically to the trusted world – achieved via the TrustZone Protection Controller (TZPC), responsible for securing on-chip peripherals, and the TrustZone Address Space Controller (TZASC) for protecting DRAM and memory-mapped devices.

3.2 TEE Challenges

TEEs currently face significant standardisation challenges [36]. For example, secure I/O remains the preserve of ARM TrustZone/GP TEE, while the GP TEE is yet to define a remote attestation mechanism. TEEs offer potential in sensing, but the absence of a uniform remote attestation solution impedes interoperability. We develop

[1]Atmel SMART: http://www.atmel.com/products/microcontrollers/ARM/default.aspx
[2]Raspberry Pi 3: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

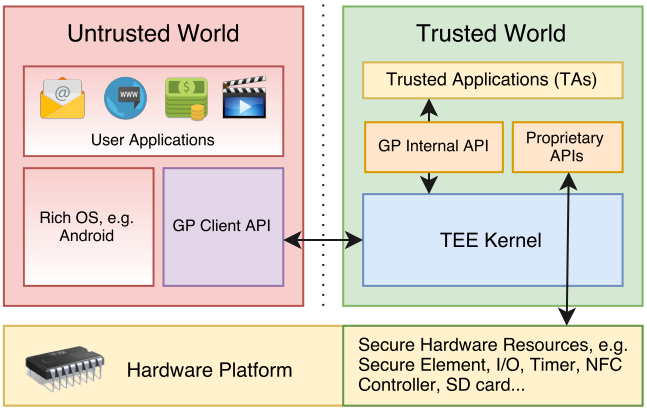

Figure 1: GlobalPlatform TEE Architecture.

the issues of remote attestation and TEE interoperability and inter-communication forthwith.

3.2.1 Remote Attestation. Intel SGX's EPID mechanism relies on developers comparing the signature of enclave applications known before deployment with the attestation result (known as a 'quote'). After receiving an attestation challenge, the untrusted host application requests its enclave to produce an attestation, and the enclave produces a report structure comprising the identity and hash of the enclave, which is verified and signed by a special quoting enclave. This is signed using a EPID key derived from a hardware-bound key, which is certified by Intel, and accessible only to the quoting enclave. The final quote is transmitted to the challenger, who can use the Intel Verification Service to verify its authenticity; this process is controlled stringently by Intel, which functions as a trusted third-party. Comparatively, GlobalPlatform is yet to standardise a remote attestation mechanism.

3.2.2 TEE Intercommunication. The issue of secure TEE-to-TEE communication on distinct devices is pertinent to sensing applications. It is conceivable that differing TEE architectures may wish to communicate with each other directly, e.g. an Intel-based server-side analytics service receiving sensitive health measurements from an ARM-based wearable. While EPID allows Intel SGX to bootstrap a secure channel, this is a one-way mechanism that only verifies an SGX-enabled entity. Ideally, both end-points should undergo trust verification to provide this assurance, i.e. each node remotely attesting the other prior to further communications involving sensitive data, and ought to be performed without trusting intermediate components, such as either REE. We herein refer to this two-way remote attestation as achieving *bi-directional* trust assurances.

To our knowledge, this remains unaddressed with remote TEEs. A naïve solution is to perform remote attestation twice – one per party – but this adds the complexity of two protocol executions, nor does it bootstrap a shared session key with two-way trust verification within a *single* run. Two-way remote attestation protocols have been proposed previously with TPMs [2, 20], and we aim to extend this to TEEs by developing a single protocol that provides intercommunication with bi-directional trust assurances, while accounting for potential interoperability issues.

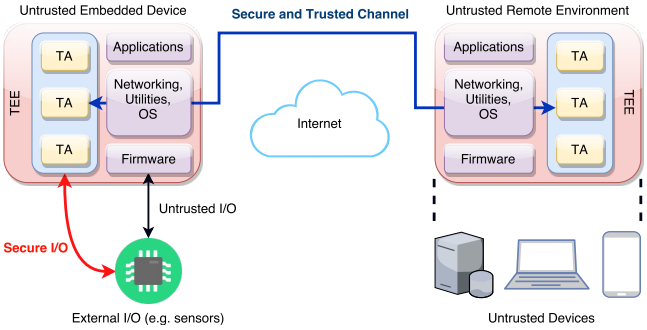

Figure 2: High-level System Architecture.

4 PROTOCOL DESIGN

We refer to the high-level architecture in Figure 2 in our discussion. The threat model assumes any component between the TEEs to be untrusted, including the medium across which data is transmitted, such as the Internet, a short-range (e.g. Bluetooth), local area, or peer-to-peer network. The TEE and its applications are considered to be trusted; the TEE should also protect against threats defined in the GlobalPlatform TEE Protection Profile [15]. We aim to defend against a strong adversary that may: **(a)**, arbitrarily observe or modify sensor measurements and other data, either in software or across a network, once it has departed either TEE; **(b)**, masquerade as either end-point with using message replays; **(c)**, attempt to use a past compromised session key in future runs; and **(d)**, deny/repudiate that a protocol instance had occurred.

4.1 Security Requirements

We propose a series of requirements for establishing trusted channels between remote TAs, which serve as a minimum (but not exhaustive) security baseline that should be supported natively:

(1) **Mutual Key Establishment**: A shared secret key is established for communication between the two entities.
(2) **Mutual Entity Authentication**: Each entity shall authenticate the other's identity to counter masquerading.
(3) **Mutual Non-Repudiation**: Neither entity may deny that a protocol instance had occurred.
(4) **Trust Assurance**: Entities shall be able to remotely attest the application and platform integrity of the other.
(5) **Mutual Trust Verification**: Both entities shall successfully attest the state of the other within the protocol.
(6) **Freshness**: The session keys, including derivatives, must be fresh in order to counter replay attacks.
(7) **Forward Secrecy**: The compromise of a particular session key should not affect past or subsequent protocol runs.
(8) **Denial of Service (DoS) Prevention**: Resource allocation shall be minimised at both end-points to prevent DoS conditions from arising.

4.2 Functional Requirements

(1) **Avoid additional trust hardware**: For constrained embedded devices, the protocol shall avoid the need for additional security hardware.

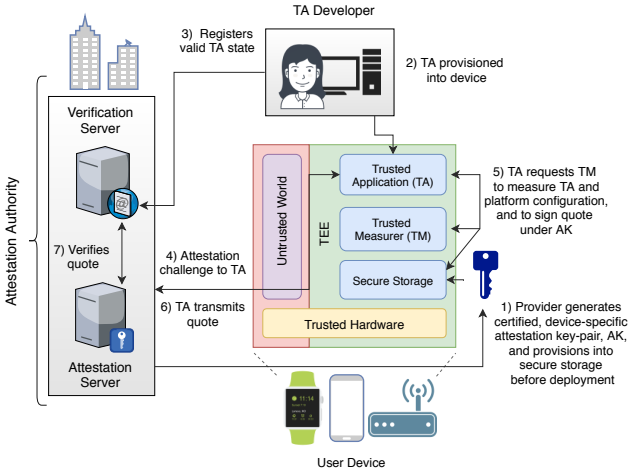

Figure 3: Generic TEE Remote Attestation Architecture.

(2) **TEE-agnostic**: The protocol shall avoid manufacturing and verification processes specific to particular TEEs.
(3) **Session resumption**: The ability to resume a session securely without restarting the protocol completely.

4.3 Trust Assurance – Operational Perspective

TEEs and TPMs typically follow the 'quote' abstraction in remote attestation. In SGX (Section 3.2.1), a separate trusted application – a quoting enclave – functions as a verifier and signatory of the state of the requested enclave and the hardware TCB. This signed state (the quote) is transmitted back to the challenger for verification by an operator, through the Intel Verification Service, which acts as a trusted authority. This abstraction is not specified in the GlobalPlatform specifications, but may be implemented similarly.

With OP-TEE [25], a GP-compliant TEE by Linaro, the TEE verifies a TA's binary signature when it is loaded into memory such that only verified TAs are launched; TAs are signed by the developer before deployment. The fact that the device was booted and the TA was launched implies a weak notion that they are integral, but this may not be fully convincing to remote challengers. This process, however, provides limited application assurances, and verifying platform integrity is more challenging. For stronger platform assurances, proprietary remote attestation mechanisms have emerged, but these remain largely undocumented publicly; Samsung KNOX, which uses ARM TrustZone, is known to transmit boot measurements as part of its attestation response, which is signed by a device-unique attestation root key certified by Samsung [37]. This is complicated by long-running TAs, like those collecting sensing data over long periods of time, which may run for days without being rebooted/reverified. A post-launch compromise by TA or TEE OS attacks outside of a TEE protection profile [15], e.g. programming errors, may not be detected except potentially by attestations that generate both TA and platform measurements (assuming such an attack does not compromise the measurer and signing keys).

We suggest the use of a separate GP TEE OS component, a Trusted Measurer (TM), which is verified in authenticate boot, to generate quotes based on the state information of the platform and

Protocol 1 Proposed Bi-directional Trust Protocol (BTP)

1: $TA_{SD} \rightarrow TA_{RE} : ID_{SD} \,||\, ID_{RE} \,||\, n_{SD} \,||\, g^{SD} \,||\, AR_{RE} \,||\, S_{cookie}$
$S_{cookie} = H(g^{SD} \,||\, n_{SD} \,||\, ID_{SD} \,||\, ID_{RE})$
2: $TA_{RE} \rightarrow TA_{SD} : ID_{RE} \,||\, ID_{SD} \,||\, n_{RE} \,||\, g^{RE} \,||\, \left[\sigma_{TA_{RE}}(X_{TA_{RE}}) \,||\, \sigma_{TA_{RE}}(V_{TA_{RE}})\right]^{K_E}_{K_{MAC}} \,||\, AR_{SD}$
$X_{TA_{RE}} = H(ID_{SD} \,||\, ID_{RE} \,||\, g^{RE} \,||\, g^{SD} \,||\, n_{RE} \,||\, n_{SD})$
$V_{TA_{RE}} = Q_{TA_{RE}} \,||\, n_{RE} \,||\, n_{SD}$
3: $TA_{SD} \rightarrow TA_{RE} : [\sigma_{TA_{SD}}(X_{TA_{SD}}) \,||\, \sigma_{TA_{SD}}(V_{TA_{SD}})]^{K_E}_{K_{MAC}} \,||\, S_{cookie}$
$X_{TA_{SD}} = H(ID_{SD} \,||\, ID_{RE} \,||\, g^{RE} \,||\, g^{SD} \,||\, n_{RE} \,||\, n_{SD})$
$V_{TA_{SD}} = Q_{TA_{SD}} \,||\, n_{RE} \,||\, n_{SD}$

Table 2: Protocol Notation

Notation	Description
SD	Sensing device, e.g. health wearable.
RE	Remote entity, e.g. analytics service.
TA_X	TEE trusted application on device X.
UA_X	Untrusted application on device X.
n_X	Random number (nonce) generated by X.
$H(D)$	A secure one-way hash function, H, applied to D.
$X \rightarrow Y$	Message transmission from X to Y.
ID_X	Identity of X.
$A \,\|\| B$	Concatenation of A and B.
g^X	Diffie-Hellman exponentiation of X.
AR_X	Attestation request on target entity X.
Q_X	Quote from TEE X.
$\sigma(A)^P_K$	Signature of A with private-public key-pair (K, P).
$[D]^{K_e}_{K_{MAC}}$	Message D is encrypted using session encryption and MAC keys K_e and K_{MAC} respectively, both generated during the protocol.

target TA. The TM would sign the quote using an attestation key provisioned by an operator before deployment. This could be in software using native TEE secure storage or, at the cost of additional hardware and Functional Requirement 1, within a secure element accessible only to the TEE to provide greater hardware tamper resistance (suggested by the GP specifications [15]). TEE implementations of TPMs ('firmware' TPMs, or fTPMs), as discussed by Raj et al. [33], are another potential candidate to act as a TM. A verification service, run by the service operator, may then verify received quotes. Figure 3 depicts a generic attestation process.

5 PROPOSED PROTOCOL

We now formalise the proposal from the requirements in Section 4 and present the Bi-directional Trust Protocol (BTP), listed in Protocol 1. In design, BTP is based on the Intel SGX attestation architecture [7], and the Greveler et al. [20] and Akram et al. [2] protocols that establish mutually trusted channels using TPMs. We provide a detailed comparison in Section 5.4. Our proposal provides bi-directional trust on TEEs while promoting interoperability through the quoting abstraction in Section 4.3. Next, we show how BTP can be modified to provide one-way attestation for devices incapable of hosting a TEE, but may still wish to transmit securely with a remote TEE. We denote this as the Uni-directional Trust Protocol (UTP); for this, the trust guarantees in Section 4 apply only to the TEE device. These differences are highlighted throughout.

5.1 Setup and Assumptions

For BTP, it is assumed both TAs know the valid state of the other, i.e. a certified signature of the TA binary, as specified in Section 3.2.1, and a TM for generating and signing quotes from application and platform state data. For UTP, only the TEE end-point must respond to attestation requests and provide quotes. It is also assumed that the TEE has a secure, preferably hardware-based, means of key generation and derivation, and random number generation.

5.2 Post-protocol

Trusted sensing necessitates secure session resumption to alleviate the overhead of re-executing the protocol. This is essential for low-powered devices that transmit sensitive data frequently but not necessarily constantly, e.g. a home monitoring system reporting occupancy data during the day. Resumption is not cost-free, however; the longer a session exists, the greater the likelihood and impact of a compromised session. As such, sessions should be re-established at regular intervals to minimise this risk.

5.3 Analysis

An informal analysis of the protocol is now discussed – marking differences between BTP and UTP – alongside the results of formal verification using Scyther.

5.3.1 Informal Discussion. **(1)**, The sensing device, which is trusted in BTP (TA_{SD}) or untrusted in UTP (UA_{SD}), computes and transmits its Diffie-Hellman (DH) exponential 'g^{SD}', along with an attestation request 'AR_{RE}' for the remote entity to provide its quote. S_{cookie}, comprising a hash of the nonce, IDs and chosen DH exponentiation may be used for session resumption. This satisfies the session resumption functional requirement (F3).

(2), Next, TA_{RE} transmits its DH exponentiation, g^{SE}; nonce, n_{RE}; its quote, $Q_{TA_{RE}}$; and signatures of both the DH exponentiations and the quote, which are signed using session-derived keys. This satisfies the requirement S1 (mutual key establishment) and S6 (freshness). In the BTP protocol, TA_{RE} requests an attestation from the sensing device. By design, this is not necessary with UTP.

(3) Finally, the sensing device acknowledges with the nonces, DH exponentiations and IDs, which are encrypted and signed, and the session cookie. This satisfies S2 (mutual entity authentication), S3 (mutual non-repudiation), S6 (freshness) and S7 (forward secrecy, by generating an ephemeral key). S8 (DoS prevention) is largely ensured by avoiding either party to perform overly-demanding operations: two signatures, a single attestation, DH exponentiation,

Table 3: Requirements Comparison of Related Protocols.

	Protocols															
Req.	STS	AD	ASPeCT	JFK	T2LS	SCP81	MM	SM	P-STCP	SSH	TLS	DTLS	SGX+EPID	GJL	AMMBSC	Proposal
S1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
S2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
S3	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
S4	✗	✗	✗	✗	✓	(✓)	✗	✗	✓	✗	✗	✗	✓	✓	✓	✓
S5	–	–	–	–	✗	✗	–	–	✗	–	–	–	✗	✓	✓	✓
S6	✓	✓	✓	✓	✓	✓	✓	(✓)	✓	✓	✓	✓	(✓)	✓	✓	✓
S7	✓	✗	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
S8	✗	✗	✗	✓	✓⋆	✗	✗	✗	✓	✗	✓⋆	✓⋆	✓	✓	✓	✓
F1	–	–	–	–	✗	✓	–	–	✗	–	–	–	•	✗	✗	✓
F2	–	–	–	–	–	–	–	–	–	–	–	–	✗	–	–	✓
F3	✗	✗	✗	✗	✓	✗	✗	✗	✓	✓⋆	✓	✓	✗	✗	✓	✓

✓— Supported. ✗— Unsupported. (✓) — Supported implicitly by the architecture. ✓⋆ — Supported after modifications. (–) — Not applicable due to another requirement. • — Keys provisioned into hardware fuses during manufacturing.

nonce generation and shared-key generation are required by each party. Lastly, the sensing device responds with its respective quote, thus satisfying S5 (bi-directional trust). For UTP, no responding quote is generated and $\sigma_{TA_{SD}}(V_{TA_{SD}})$ is not transmitted.

5.3.2 Mechanical Formal Verification with Scyther. The Scyther verification tool by Cremers et al. [10] was used to verify the correctness of both protocols. Given a security protocol description written in the Scyther description language and desired properties (claims), Scyther verifies whether the protocol satisfies those claims. We analyse both protocols using the Dolev-Yao adversarial model, testing for the secrecy of quotes (`Secret, qre`), aliveness (`Alive`), non-injective agreement (`Niagree`), non-injective synchronisation (`Nisynch`), session key secrecy (`SKR, K`) and reachability of both communicating entities (`TARE or UASD/TASD, Reachable`). The full scripts for the bi-directional (BTP) and uni-directional (UTP) protocols can be found in Appendices A.1 and A.2 respectively. Scyther found no attacks on either protocol.

5.4 Comparison

A criteria comparison is made with protocols from related domains in Table 3. These include well-known transport-layer protocols (TLS [11] and SSH [47]), and their variants (DTLS [35]), as well as smart cards (the Markantonakis-Mayes [27], GP SCP81 [16] and Sirett-Mayes [45] protocols). We also include other key exchange (Station-to-Station [12], Aziz-Diffie [6], ASPeCT [23] and Just Fast Keying or JFK [1]) and trusted protocols (Trusted TLS or T2LS [13], Akram et al. (AMMBSC) [2], Greveler et al. (GJL) [20], Enhanced Privacy ID with SGX [7] and P-STCP [3]. The smart card domain offers an unique perspective, with high security and performance requirements, as well as its maturity, and we include such protocols due to the similar requirements on limited devices.

As illustrated in Table 3, no protocol satisfies the requirements motivating this work. Some are closely related, such as AMMBSC [2], SGX+EPID [7] and GJL [20]. SGX+EPID is not considered due to the tightly-controlled nature in which Intel provisions PSKs during manufacturing and performs attestation (described in Section 3.2.1). Naturally, AMMBSC is avoided for its reliance on TPM, the issues of which we highlighted in Section 3. GJL, which is similar to AMMBSC and uses TPMs, is avoided likewise. While widely used, (D)TLS and SSH are avoided due to the absence of trust provisions, alongside the difficulty of engineering TEE-based attestation in such protocols retroactively.

6 IMPLEMENTATION

Both BTP and UTP were implemented using OP-TEE [25] – an open-source, GP-compliant TEE based on ARM TrustZone. Two applications were developed that executed in the trusted and untrusted worlds independently, with communications mediated via the GP secure monitor [19]. OP-TEE executes two execution environments simultaneously: the untrusted world OP-TEE Client, running a bare-bones Linux system that implements the GP TEE Client API [17]; and OP-TEE OS, running the TEE kernel that implements the GP TEE Internal API [18]. The reader is referred back to Section 3 and Figure 1 for an overview of the GP TEE.

Both protocols were implemented using the cryptographic operations defined in the GP Internal API, all of which occurred in the trusted world application, while the untrusted world was used for handling TCP/IP sockets and TA instantiation. In OP-TEE, cryptographic operations are implemented by the LibTomCrypt library and ARM Cryptography Extensions for providing instruction-level AES and SHA [4]. Based on the NIST recommendations [30], we used 2048-bit Diffie-Hellman, 128-bit AES in CBC mode, and SHA-256 (including for HMACs). For quote signing and verification, we used the Elliptic Curve Digital Signature Algorithm (ECDSA) using the `secp256r1` curve, based on those recommendations. Currently, only the NIST curves are specified in the GP TEE Internal API [18]. Each TA was preloaded with the known signature of the remote TA and an associated 256-bit ECDSA public key, also using `secp256r1`. Akin to Intel SGX enclave measurement, we emulated a TM that, on request, returns an ECDSA-signed hash of the binary and mock platform information. In practice, as per Section 3, this key would be certified and provisioned by the developer, and the TM would ideally form part of the TEE's secure boot.

We used two HiKey LeMaker ARM development boards, hosting an HiSilicon Kirin 620 SoC with an eight-core ARM Cortex-A53 processor (1.2GHz), 2GB DDR3 RAM and TrustZone extensions. Such specifications are typical of modern single-board platforms;

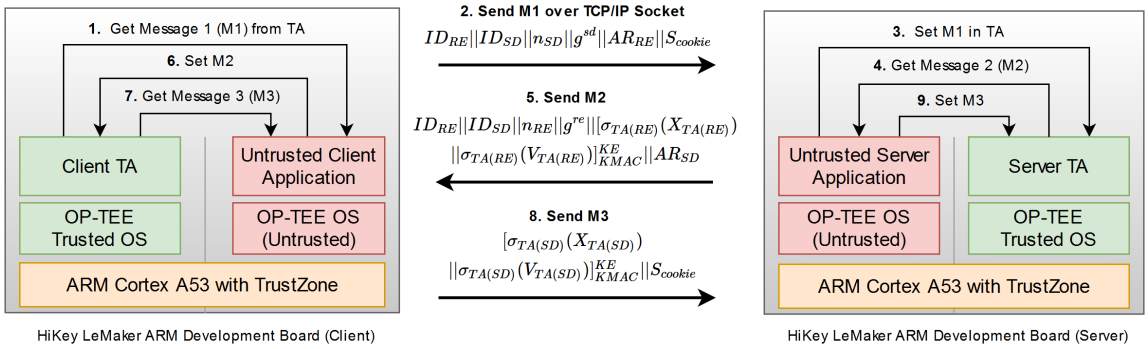

Figure 4: High-level Implementation Information Flow for BTP.

the Raspberry Pi 3, for example, also uses an ARM Cortex-A53. Notably, the board is supported and well-documented for use with OP-TEE. The protocol was implemented in C – the only development language supported fully by OP-TEE at the time of writing – and cross-compiled for ARM 64-bit platforms using the GNU C Compiler. In order to minimise overhead, world context switches were kept to two per message for setting and retrieving messages into and from the TA respectively. This is shown in Figure 4.

7 PERFORMANCE EVALUATION

We measured 1,000 runs of the BTP and UTP protocols, which were benchmarked against the OpenSSL 1.0.1 implementation of TLS v1.2 and SSH (via OpenSSH 6.7) running in the untrusted world. For TLS, the modes `DHE-RSA-AES128-SHA256`, `DHE-DSS-AES128-SHA256` and `ECDHE-ECDSA-AES128-SHA256` were used. This selection stems from the similarities with our proposal: the use of ephemeral Diffie-Hellman (DHE), 256-bit ECDSA and 128-bit AES, in addition to RSA and DSS signatures for comparison. These were measured using `openssl s_client`. For SSH, 128-bit AES in CBC mode was used with SHA-256 HMACs, as per our implementation. This was tested with RSA-, ECDSA- and DSA-based key pairs; the `ssh` utility was used with default settings except for specifying AES and HMAC modes, and the desired key. The Diffie-Hellman group sizes were set at 2048 bits (with pre-generated moduli) in accordance with our implementation, as well as 256-bit ECDSA for the relevant SSH and TLS modes; 3072-bit RSA was also used, in line with the NIST recommendations. Note that 3072-bit DSS was used for TLS, while OpenSSH imposes a 1024-bit limit for DSA; OpenSSH aims to deprecate DSA, but we include it for comparison. The protocols were measured using wall-clock time via the `time` UNIX utility and assisted by a shell script for initialisation, storing the target IP address and port number, and result logging.

Message-specific times were measured for BTP and UTP using the `<time.h>` library provided by OP-TEE. For these, the mean wall-clock time was calculated from 100 measurements per message. This includes client and server times to generate and verify each message, and round-trip time to account for network latency – comprising message generation, transmission, verification and acknowledgement. For all experiments, the boards were connected through an Ethernet hub with no other attached devices to minimise network overhead. The boards were configured in a client-server architecture: one acting as TA_{RE} and the other as TA_{SD}. Both were connected to a laptop via UART-to-USB for debugging purposes. Figure 5 depicts our test-bed environment.

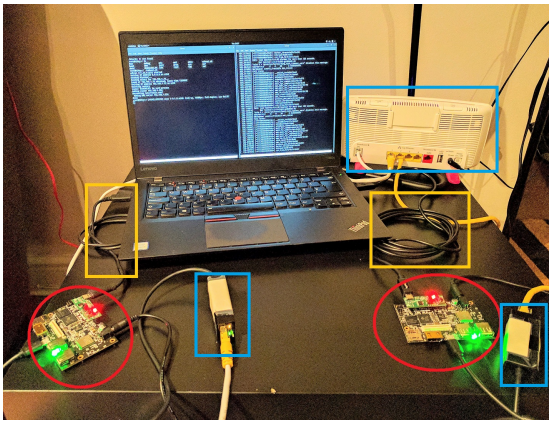

Figure 5: Test-bed Environment – HiKey Boards (circled) Connected via Ethernet (blue) and UART (yellow).

7.1 Results Discussion

The results are presented in Figure 6 (comparing BTP and UTP with TLS and SSL), Table 4 (BTP and UTP message-specific times), and Table 5 (total protocol round-trip time). As shown, our proposal performs consistently at approximately **1.7** seconds to execute in its entirety (round-trip) – yielding a $\tilde{4}$x overheard versus TLS with ephemeral key Diffie-Hellman and RSA. This comprises the time to open/close TCP/IP sockets, receive and parse messages from the client/server, invoking the protocol TA functions and world context switches. This is in addition to performing all of the cryptographic algorithms including 2048-bit Diffie-Hellman key-pair generation, HMAC and signature generation/verification (including for quotes).

Some overhead was expected due to the results from related TPM and TEE work [2, 26, 41], and we attribute this to: **(1)**, the difference in cryptographic implementations between SSH and TLS, both of which used OpenSSL, and the less widely-used LibTomCrypt provided by OP-TEE; **(2)**, the penalty imposed by two world context switches per message and instantiation/destruction of the TA; **(3)**, additional overhead of packing and unpacking message

Table 4: Mean Client, Server and Round-trip Wall-clock Times in Milliseconds (Std. Dev. in Brackets).

Message	BTP Client (TA_{SD})	BTP Server (TA_{RE})	BTP Round-trip	UTP Client (UA_{SD})	UTP Server (TA_{RE})	UTP Round-trip
M1: $TA/UA_{SD} \rightarrow TA_{RE}$	**302.7** (4.1)	**21.5** (2.9)	**347.1** (3.5)	**305.9** (2.5)	**20.8** (3.0)	**350.2** (2.6)
M2: $TA_{RE} \rightarrow TA/UA_{SD}$	**346.9** (6.6)	**682.6** (6.9)	**1192.0** (7.2)	**350.3** (6.1)	**689.4** (7.3)	**1189.2** (10.3)
M3: $TA/UA_{SD} \rightarrow TA_{RE}$	**61.6** (4.8)	**39.7** (4.1)	**117.2** (4.6)	**39.2** (3.9)	**26.7** (3.1)	**73.8** (3.4)

Table 5: Mean Full Protocol Wall-clock Times in Milliseconds.

Proposal		TLS			SSH		
BTP	UTP	DHE+RSA	DHE+DSS	ECDHE+ECDSA	RSA	DSA	ECDSA
1692.3 (11.0)	**1618.2** (9.6)	**410.5** (3.9)	**374.3** (4.1)	**102.5** (2.2)	**535.1** (13.6)	**446.1** (11.8)	**453.8** (15.0)

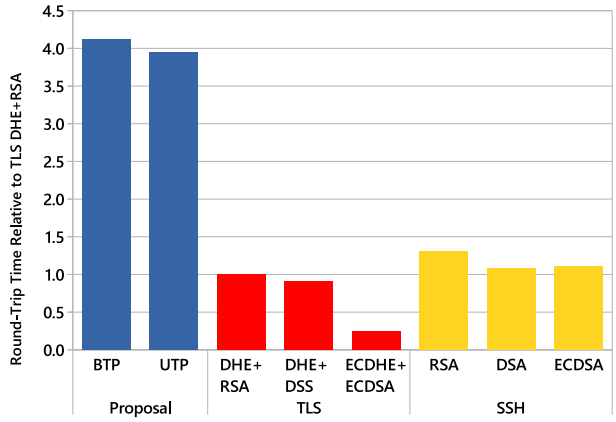

Figure 6: Protocol Performance Relative to TLS (DHE-RSA).

structures to adhere to GP-specific data structures in the Client [15] and Internal APIs [18].

Further, our reference implementation could benefit from optimisations, which is usual with widely-used protocol implementations. There is an approximate 100ms overhead of the full protocol time versus message totals. We attribute this specifically to the initialisation and tear-down of the TA, i.e. **(1)** and **(2)**. Notably, a significant portion of the protocol time occurs where Diffie-Hellman operations are present. This occurs once on the client-side in message 1 and three times in message 2: twice server-side – once to compute g^{RE} and the other to compute the shared secret – and another on the client-side to compute its shared value. We refer back to **(3)** for a potential explanation. While benchmarking GlobalPlatform primitives and cryptographic suites is outside the scope of this paper, we identified the above two as two major performance bottlenecks.

7.2 Limitations

TEEs offer their own benefits in trusted sensing, but avoiding TPMs relinquishes extensive hardware tamper-resistance. TEEs do not necessarily defend against sophisticated hardware attacks and side-channel analyses, but are designed primarily to resist software-based vectors [9, 19, 36]. Consequently, we urge caution in using our work where complex hardware attacks are a reasonable possibility, e.g. military and government applications, where state-level adversaries comprise part of a threat model. Secondly, GlobalPlatform recently defined the Sockets API for initiating TCP/IP connections from the trusted world; this provides an interesting avenue of research, which, in the context of our protocol, would remove a context switch between the secure and trusted world. Mature implementations of the Sockets API are not widely available for development boards; with OP-TEE, for example, GP Sockets API remains in its infancy, and we aim to revisit this in the future upon maturity. Finally, defending against availability attacks is currently an open challenge in TEE research [36]. TEEs rely implicitly upon co-operation between untrusted and trusted world, and a compromised untrusted world that simply drops messages from the TEE would prevent the protocol from running, thus raising availability concerns; this issue, however, is inherent across TEEs.

8 CONCLUSION

In this work, we investigated the application of TEEs to trusted sensing applications, and identified current challenges facing their deployment, namely secure TEE intercommunication. We contribute to this by presenting the development and evaluation of a secure and trusted channel protocol that, unlike past work, provides one- or two-way remote attestation for trust assurance while limiting the trust placed in intermediary components. The proposal benefits from enabling isolated execution, secure I/O, and communication of sensitive sensing data in a manner that defends against a compromised REE. In Sections 4 and 5, both the requirements and protocols were formalised, along with the operational challenges. In Section 6, we discussed protocol implementation using two widely-used development boards using ARM TrustZone, before evaluating its performance against untrusted world TLS and SSH in Section 7. Both protocols were subjected to formal analysis using Scyther, which found no attacks. We showed that our proposal executes within in reasonable time (under 1.7 seconds on average) and exhibits approximately 4x overhead versus TLS and SSH.

8.1 Future Research

In future work, we aim to investigate the following lines of research:

- *Trusted multi-party protocol.* Generalising our protocol to create a trusted channel shared between multiple devices in close proximity. This could be advantageous where security-sensitive, multi-party communications are used frequently, such as authentication schemes using sensor data from a body area network (BAN), or a sensor-driven home automation system.
- *Performance comparison with TPMs and other TEEs.* We aim to conduct a broad performance evaluation of our protocol with TPM-based solutions and emerging TEEs, such as the AMD Platform Security Processor and Trustonic Kinibi, on a range of device types. Furthermore, we plan a performance comparison with other cryptographic libraries.
- *Applicability of elliptic curves.* Recent GP specifications include Elliptic Curve Cryptography (ECC), but this is only an optional extra for compliance. Moreover, only five NIST curves are supported currently, an further curves are expected to be released in the future [18]. Once matured, we aim to revisit our protocol to provide elliptic curve Diffie-Hellman to reduce key sizes and computational overhead, and evaluating the range of curves on offer.

ACKNOWLEDGMENTS

Carlton Shepherd is supported by the EPSRC and the British government as part of the Centre for Doctoral Training in Cyber Security at Royal Holloway, University of London (EP/K035584/1). The authors would like to thank the reviewers for their insightful comments.

REFERENCES

[1] William Aiello, Steven M Bellovin, Matt Blaze, Ran Canetti, John Ioannidis, Angelos D Keromytis, and Omer Reingold. 2004. Just fast keying: Key agreement in a hostile internet. *ACM Transactions on Information and System Security (TISSEC)* 7, 2 (2004), 242–273.

[2] Raja Akram, Konstantinos Markantonakis, Keith Mayes, Pierre-Francois Bonnefoi, Damien Sauveron, and Serge Chaumette. 2016. *An Efficient, Secure and Trusted Channel Protocol for Avionics Wireless Networks.* IEEE Computer Society.

[3] Raja Naeem Akram, Konstantinos Markantonakis, and Keith Mayes. 2012. A privacy preserving application acquisition protocol. In *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications.* IEEE, 383–392.

[4] ARM. 2015. ARM Cortex Programmer's Guide for ARMv8-A. Version 1.0. (2015). http://infocenter.arm.com/help/topic/com.arm.doc.den0024a/DEN0024A_v8_architecture_PG.pdf.

[5] Myo Min Aung and Yoon Seok Chang. 2014. Temperature management for the quality assurance of a perishable food supply chain. *Food Control* 40 (2014), 198–207.

[6] Ashar Aziz and Whitfield Diffie. 1994. Privacy and authentication for wireless local area networks. *IEEE Personal Communications* 1, 1 (1994), 25–31.

[7] Ernie Brickell and Jiangtao Li. 2011. Enhanced privacy ID from bilinear pairing for hardware authentication and attestation. *International Journal of Information Privacy, Security and Integrity 2* 1, 1 (2011), 3–33.

[8] Capgemini. 2016. Wearable Devices and their Applicability in the Life Insurance Industry. (2016). https://www.capgemini.com/resource-file-access/resource/pdf/wearable_devices_and_their_applicability_in_the_life_insurance_industry.pdf.

[9] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* 2016 (2016), 86. https://eprint.iacr.org/2016/086.pdf.

[10] Cas JF Cremers. 2008. The Scyther Tool: Verification, falsification, and analysis of security protocols. In *International Conference on Computer Aided Verification.* Springer, 414–418.

[11] Tim Dierks and Eric Rescorla. 2008. RFC 5246 – Transport Layer Security (TLS) Protocol Version 1.2. (August 2008). https://tools.ietf.org/html/rfc5246.

[12] Whitfield Diffie, Paul C Van Oorschot, and Michael J Wiener. 1992. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography* 2, 2 (1992), 107–125.

[13] Yacine Gasmi, Ahmad-Reza Sadeghi, Patrick Stewin, Martin Unger, and N. Asokan. 2007. Beyond Secure Channels. In *Proceedings of the 2007 ACM Workshop on Scalable Trusted Computing (STC '07).* ACM, New York, NY, USA, 30–40. DOI: http://dx.doi.org/10.1145/1314354.1314363

[14] Peter Gilbert, Landon P Cox, Jaeyeon Jung, and David Wetherall. 2010. Toward trustworthy mobile sensing. In *Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications.* ACM, 31–36.

[15] GlobalPlatform. 2014. TEE Protection Profile (Version 1.2). (2014).

[16] GlobalPlatform. 2015. Card Remote Application Management over HTTP. (2015).

[17] GlobalPlatform. 2016. GlobalPlatform TEE Client API Specification v1.0. (2016).

[18] GlobalPlatform. 2016. GlobalPlatform TEE Internal API Specification v1.0. (2016).

[19] GlobalPlatform. 2016. GlobalPlatform TEE System Architecture Specification v1.1. (2016).

[20] Ulrich Greveler, Benjamin Justus, and Dennis Loehr. 2011. Mutual remote attestation: enabling system cloning for TPM based platforms. In *International Workshop on Security and Trust Management.* Springer, 193–206.

[21] Ramakrishna Gummadi, Hari Balakrishnan, Petros Maniatis, and Sylvia Ratnasamy. 2009. Not-a-Bot: Improving Service Availability in the Face of Botnet Attacks.. In *NSDI*, Vol. 9. 307–320.

[22] Dae-Man Han and Jae-Hyun Lim. 2010. Smart home energy management system using IEEE 802.15.4 and ZigBee. *IEEE Transactions on Consumer Electronics* 56, 3 (2010).

[23] Günther Horn and Bart Preneel. 2000. Authentication and Payment in Future Mobile Systems. *Journal of Computer Security* 8, 2,3 (Aug. 2000), 183–207. http://dl.acm.org/citation.cfm?id=1297828.1297832

[24] Xiaochun Li, Anastasios Golnas, and Fritz B Prinz. 2000. Shape deposition manufacturing of smart metallic structures with embedded sensors. In *SPIE's 7th Annual International Symposium on Smart Structures and Materials.* International Society for Optics and Photonics, 160–171.

[25] Linaro. 2017. OP-TEE : Open Source Trusted Execution Environment. (2017). https://www.op-tee.org/.

[26] He Liu, Stefan Saroiu, Alec Wolman, and Himanshu Raj. 2012. Software abstractions for trusted sensors. In *Proceedings of the 10th international conference on Mobile systems, applications, and services.* ACM, 365–378.

[27] Konstantinos Markantonakis and Keith Mayes. 2005. A Secure Channel protocol for multi-application smart cards based on public key cryptography. In *Communications and Multimedia Security.* Springer, 79–95.

[28] Jonathan M McCune, Bryan J Parno, Adrian Perrig, Michael K Reiter, and Hiroshi Isozaki. 2008. Flicker: An execution infrastructure for TCB minimization. In *ACM SIGOPS Operating Systems Review*, Vol. 42. ACM, 315–328.

[29] Esmaeil S Nadimi, Rasmus Nyholm Jørgensen, Victoria Blanes-Vidal, and Svend Christensen. 2012. Monitoring and classifying animal behavior using ZigBee-based mobile ad hoc wireless sensor networks and artificial neural networks. *Computers and Electronics in Agriculture* 82 (2012), 44–54.

[30] NIST. 2016. Recommendations for Key Management. (2016). Special Publication 800-57 Part 1 Rev. 4.

[31] Norbert Noury, Thierry Hervé, Vicent Rialle, Gilles Virone, Eric Mercier, Gilles Morey, Aldo Moro, and Thierry Porcheron. 2000. Monitoring behavior in home using a smart fall sensor and position sensors. In *Microtechnologies in Medicine and Biology, 1st Annual International, Conference On. 2000.* IEEE, 607–610.

[32] Alexandros Pantelopoulos and Nikolaos G Bourbakis. 2010. A survey on wearable sensor-based systems for health monitoring and prognosis. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 40, 1 (2010), 1–12.

[33] Himanshu Raj, Stefan Saroiu, Alec Wolman, Ronald Aigner, Jeremiah Cox, Paul England, Chris Fenner, Kinshuman Kinshumann, Jork Loeser, Dennis Mattoon, Magnus Nystrom, David Robinson, Rob Spiger, Stefan Thom, and David Wooten. 2016. fTPM: A Software-Only Implementation of a TPM Chip. In *Proceedings of the 25th USENIX Security Symposium (USENIX Security 16).* USENIX Association, Austin, TX, 841–856. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/raj

[34] Parisa Rashidi and Alex Mihailidis. 2013. A survey on ambient-assisted living tools for older adults. *IEEE journal of biomedical and health informatics* 17, 3 (2013), 579–590.

[35] E. Rescorla and N. Modadugu. 2012. RFC 6347 – Datagram Transport Layer Security (DTLS) Version 1.2. (January 2012). https://tools.ietf.org/html/rfc6347.

[36] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. 2015. Trusted execution environment: what it is, and what it is not. In *Trustcom/BigDataSE/ISPA, 2015 IEEE*, Vol. 1. IEEE, 57–64.

[37] Samsung Electronics Co. 2015. An Overview of the Samsung KNOX Platform. (November 2015). http://www.samsung.com/global/business/business-images/resource/white-paper/2013/06/Samsung_KNOX_whitepaper_June-0.pdf.

[38] Nuno Santos, Himanshu Raj, Stefan Saroiu, and Alec Wolman. 2011. Trusted language runtime (TLR): enabling trusted applications on smartphones. In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications.* ACM, 21–26.

[39] A. Sarela, I. Korhonen, J. Salminen, E. Koskinen, O. Kirkeby, and D. Walters. 2009. A home-based care model for outpatient cardiac rehabilitation based on mobile technologies. In *2009 3rd International Conference on Pervasive Computing Technologies for Healthcare.* 1–8. DOI: http://dx.doi.org/10.4108/ICST.

PERVASIVEHEALTH2009.5970

[40] Stefan Saroiu and Alec Wolman. 2010. I am a sensor, and I approve this message. In *Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications*. ACM, 37–42.

[41] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich. VC3: Trustworthy Data Analytics in the Cloud Using SGX. In *2015 IEEE Symposium on Security and Privacy*.

[42] Carlton Shepherd, Raja N Akram, and Konstantinos Markantonakis. 2017. Towards Trusted Execution of Continuous Authentication Schemes. In *Proceedings of the 32nd ACM Symposium on Applied Computing*. ACM.

[43] Carlton Shepherd, Ghada Arfaoui, Iakovos Gurulian, Robert P Lee, Konstantinos Markantonakis, Raja N Akram, Damien Saveron, and Emmanuel Conchon. 2016. Secure and Trusted Execution: Past, Present, and Future - A Critical Review in the Context of the Internet of Things and Cyber-Physical Systems. In *2016 IEEE Trustcom/BigDataSE/ISPA*. 168–177. DOI: http://dx.doi.org/10.1109/TrustCom.2016.0060

[44] Weidong Shi, Jun Yang, Yifei Jiang, Feng Yang, and Yingen Xiong. 2011. Senguard: Passive user identification on smartphones using multiple sensors. In *Wireless and Mobile Computing, Networking and Communications (WiMob), 2011 IEEE 7th International Conference on*. IEEE, 141–148.

[45] William G Sirett, John A MacDonald, Keith Mayes, and Konstantinos Markantonakis. 2006. Design, installation and execution of a security agent for mobile stations. In *International Conference on Smart Card Research and Advanced Applications*. Springer, 1–15.

[46] Upkar Varshney. 2007. Pervasive healthcare and wireless health monitoring. *Mobile Networks and Applications* 12, 2-3 (2007), 113–127.

[47] Tatu Ylonen and Chris Lonvick. 2006. RFC 4253 – The Secure Shell (SSH) Transport Layer Protocol. (January 2006). https://tools.ietf.org/html/rfc4253.

[48] Zongwei Zhou, Virgil D Gligor, James Newsome, and Jonathan M McCune. 2012. Building verifiable trusted path on commodity x86 computers. In *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE, 616–630.

A APPENDIX

A.1 BTP Protocol Scyther Source

```
hashfunction h;
// Attestation Request
const AttReq: Function;
// Attestation response (quote)
usertype Quote;
usertype SessionKey;
// Diffie-Hellman components
usertype DH;

protocol btp(TASD,TARE)
{
  macro Scookie = h(gSD,nsd,TASD,TARE);
  role TASD
  {
    fresh nsd: Nonce;
    var nre: Nonce;
    fresh qsd: Quote;
    var qre: Quote;
    var K: SessionKey;
    fresh gSD: DH;
    var gRE: DH;

    send_1(TASD,TARE, nsd, gSD, AttReq, Scookie);
    recv_2(TARE, TASD, nre, gRE,{{TARE,TASD,nre,nsd,gSD,gRE}sk(
         TARE),{qre, nre, nsd}sk(TARE)}K, AttReq);
    send_3(TASD,TARE,{{h(TASD,TARE,gRE,gSD,nre,nsd)}sk(TASD),{qsd,
         nre,nsd}sk(TASD)}K, Scookie);

    claim_a1(TASD, Alive);
    claim_a2(TASD, Niagree);
    claim_a3(TASD, Nisynch);
    claim_a4(TASD, Secret, qsd);
    claim_a5(TASD, Reachable);
    claim_a6(TASD, SKR, K);
  }

  role TARE
  {
    fresh nre: Nonce;
    var nsd: Nonce;
    fresh qre: Quote;
    var qsd: Quote;
    fresh K: SessionKey;
    fresh gRE: DH;
    var gSD: DH;

    recv_1(TASD, TARE, nsd, gSD, AttReq, Scookie);
    send_2(TARE, TASD, nre, gRE,{{TARE,TASD,nre,nsd,gSD,gRE}sk(
         TARE),{qre, nre, nsd}sk(TARE)}K, AttReq);
    recv_3(TASD,TARE,{{h(TASD,TARE,gRE, gSD, nre,nsd)}sk(TASD), {
         qsd,nre,nsd}sk(TASD)}K, Scookie);

    claim_b1(TARE, Alive);
    claim_b2(TARE, Niagree);
    claim_b3(TARE, Nisynch);
    claim_b4(TARE, Secret, qre);
    claim_b5(TARE, Reachable);
    claim_b6(TARE, SKR, K);
  }
}
```

A.2 UTP Protocol Scyther Source

```
protocol utp(UASD,TARE)
{
  macro Scookie = h(gSD,nsd,UASD,TARE);
  role UASD
  {
    fresh nsd: Nonce;
    var nre: Nonce;
    var qre: Quote;
    var K: SessionKey;
    fresh gSD: DH;
    var gRE: DH;

    send_1(UASD,TARE, nsd, gSD, AttReq, Scookie);
    recv_2(TARE, UASD, nre,  gRE,{{TARE,UASD,nre,nsd,gSD,gRE}sk(
         TARE),{qre,nre,nsd}sk(TARE)}K);
    send_3(UASD,TARE,{{h(UASD,TARE,gRE,gSD,nre,nsd)}sk(UASD)}K,
         Scookie);

    claim_a1(UASD, Alive);
    claim_a2(UASD, Niagree);
    claim_a3(UASD, Nisynch);
    claim_a5(UASD, Reachable);
    claim_a6(UASD, SKR, K);
  }

  role TARE
  {
    fresh nre: Nonce;
    var nsd: Nonce;
    fresh qre: Quote;
    fresh K: SessionKey;
    fresh gRE: DH;
    var gSD: DH;

    recv_1(UASD, TARE, nsd, gSD, AttReq, Scookie);
    send_2(TARE, UASD, nre, gRE,{{TARE,UASD,nre,nsd,gSD,gRE}sk(
         TARE), {qre,nre,nsd}sk(TARE)}K);
    recv_3(UASD,TARE,{{h(UASD,TARE,gRE, gSD,nre,nsd)}sk(UASD)}K,
         Scookie);

    claim_b1(TARE, Alive);
    claim_b2(TARE, Niagree);
    claim_b3(TARE, Nisynch);
    claim_b4(TARE, Secret, qre);
    claim_b5(TARE, Reachable);
    claim_b6(TARE, SKR, K);
  }
}
```