Anonymous Trusted Data Relocation for TEEs

Vasco Guita, Daniel Andrade, João Nuno Silva, Miguel Correia

INESC-ID, Instituto Superior Técnico, Universidade de Lisboa – Lisboa, Portugal
{vasco.guita,daniel.andrade,joao.n.silva,miguel.p.correia}@tecnico.ulisboa.pt

Abstract. Trusted Execution Environment (TEE) technology like ARM TrustZone allows protecting confidential data using cryptographic keys that are bound to a specific TEE and device. However, there are good reasons to allow relocating such data from a TEE to another TEE in another device, often in a non-interactive (offline) and anonymous manner. We propose the Trusted Relocation Extension (TRX), a TrustZone-based trusted storage service enabling backup/recovery and sharing of data between TEEs in different devices. TRX works offline, without previous key exchange, and ensures the anonymity of the sender and the receiver. We present an implementation of TRX compatible with OP-TEE and its evaluation with Raspberry Pi 3 B+ devices.

1 Introduction

Trusted Execution Environment (TEE) technology has the important role of reducing the Trusted Computing Base (TCB) and the attack surface of services or functions that run inside the TEE itself [26]. The technology of this type we consider in the paper is ARM TrustZone, a security extension of ARM processors that supports two separate environments: the normal world, that runs the Rich Execution Environment (REE) software stack; and the secure world, that runs the TEE software. TrustZone provides hardware-enforced isolation to the secure world, guaranteeing that the REE has no access to the memory and resources of the TEE. Something similar is offered by the Intel Software Guard Extensions (SGX) [24] and Sanctum [11], among others.

A service that can be provided by a TEE is *trusted storage*, used for storing private and confidential data, since TEEs can isolate data from the REE [17]. This protection is enforced with *cryptographic keys bound to the specific TEE and device that stores the data.* Current TrustZone trusted storage systems depend on the Hardware Unique Key (HUK) for deterministically deriving the Secure Storage Key (SSK) to protect their data. The HUK is a Root of Trust (RoT) element that is written by the manufacturer on the device's One Time Programmable (OTP) memory guaranteeing that trusted storage data is bound to the device that encrypts it. Although positive from the security angle, this binding can also be seen as a limitation.

There are good reasons to allow the *relocation of encrypted data from a TEE to a different TEE in another device.* One is when applications require sharing data with another person or system. Another is performing data backups inside the TEE and recover them later in another device, as the devices can be stolen or get broken. In both cases, data transfer is often done *offline*, i.e., in a non-interactive manner. For example, data is transferred first to a flash drive or an external disk, and only then, and when needed, to the destination device. This avoids the constraint of having the destination device connected and available when the data is transferred, which is desirable when doing backups. In addition, in several applications there is a requirement of anonymity of the sender and/or the receiver. For example, during a data transfer from a clinic to a patient, we may not want the clinic specialized in cancer to reveal that it is the source of a personal health report and who is the receiver.

The paper presents the design of the *Trusted Relocation Extension (TRX)*, a TrustZone-based trusted storage service that allows sharing and backing-up/recovering data between/in TEEs in different devices. The design goals of TRX are: (1) *DG1 Non-interactivity* – data can be transferred offline, i.e., without the need of an online handshake between source and destination TEEs, either as part of the transfer or at some initial point in time for key exchange; (2) *DG2 Anonymity* – the identities of the sender and the receiver are not disclosed in the transference; (3) *DG3 Confidentiality* – no entity other than the sender and the receiver can read the data; (4) *DG4 Integrity* – the receiver can verify if the data transferred was modified in the transference.

A few classical solutions to this problem come immediately to mind: (1) to encrypt and export the SSK using public-key cryptography (e.g., RSA or ECC) and use this key to ensure confidentiality (DG3) and integrity (DG4), but this requires the distribution of public keys in certificates, breaking non-interactivity (DG1) and possibly anonymity (DG2); (2) to encrypt and export the SSK with a password-derived key, but this requires users to memorize and share passwords, which is inconvenient and breaks non-interactivity (DG1) and anonymity (DG2); (3) to display a QR code with the SSK in a TEE-controlled display and scan it with a TEE-controlled camera, but this would bloat the TCB with much image handling software and break non-interactivity (DG1) and anonymity (DG2).

With the TRX service, each data bundle, called Trusted Volume (TV), is protected with its own secret key, the Trusted Volume Key (TVK). TRX uses a very recent cryptographic scheme, *Matchmaking Encryption* [5], to encrypt the TVK when relocating a TV. Matchmaking Encryption allows protecting non-interactive communication between two entities. It allows the sender and the receiver to impose policies that the other party has to satisfy in order to reveal the message, TVK in our case.

We implemented a prototype of the TRX service for OP-TEE that runs as a companion to Linux on a Raspberry Pi (RPI) 3 Model B+.[1] Our work is one of the first to use Matchmaking Encryption [9,33] and the first to use Matchmaking Encryption with ARM TrustZone. We implemented our own version of Identity-based Matchmaking Encryption (IB-ME) as a C library (the first IB-ME library) to avoid bloating the TCB of TRX with the dependencies of the existing IB-ME

[1] All software available at: https://github.com/vascoguita/trx

prototype [5] which is implemented in Python. Our evaluation shows that the TCB overhead and time costs involved are small.

The main contributions of our work are: (1) TRX, a new solution to relocate encrypted data from a TEE in a device to another TEE in a different device, with non-interactivity (DG1), anonymity (DG2), confidentiality (DG3), and integrity (DG4), which is also one of the first schemes to leverage IB-ME; (2) the conceptual instantiation of TRX in OP-TEE and TrustZone; (3) the experimental evaluation of TRX in Raspberry Pi 3 Model B+ devices; (4) the first IB-ME library (in C); (5) a prototype of the TRX service, and a Proof of Concept (PoC) Trusted Authority that supports this mechanism.

2 Background

2.1 ARM TrustZone

A physical core of a TrustZone-enabled processor is divided into two virtual cores [2]: the normal world that hosts the REE, where the main Operating System (OS) and the untrusted applications run; the secure world that hosts the TEE, where the trusted software runs. These worlds are isolated from each other, with independent memory address spaces and different privileges, in such a way that applications running in the secure world can access memory regions associated with the normal world, but not the opposite [26]. The context switch between worlds is handled by the *Secure Monitor* (that runs in Monitor Mode) that is responsible for preserving the state of the current world and restoring the state of the world being switched to; and the processor that switchs from the normal world to the secure world when some process calls the Secure Monitor Call (SMC) instruction or a hardware exception is raised.

In a TrustZone-enabled device, the DRAM is partitioned into *Secure* and *Non-Secure* regions, with the assistance of the TrustZone Address Space Controller (TZASC). Peripherals can be reserved for exclusive secure world use by the TrustZone Protection Controller (TZPC). This feature can be used to implement a Trusted User Interface (TUI) [21] to allow users to interact directy with the TEE.

Trusted Applications (TAs), also called truslets [26, 28], are programs that perform sensitive operations in the TEE, therefore isolated from the REE. REE applications perform requests to a TA using the TEE Client API. This API puts the request in a message buffer and calls the SMC instruction, which passes the control to the secure world. Then, the trusted kernel invokes the requested TA, which takes the request from the message buffer. After executing the requested operation, the TA responds to the REE application using the same buffer.

The trustworthiness of a TrustZone-based TEE software stack depends on its Chain of Trust (CoT) [4]. The CoT starts with two implicitly trusted components of the Trusted Board Boot (TBB) sequence [3]: (1) The hash of the Root of Trust Public Key (ROTPK), stored on the System-on-a-Chip (SoC)'s OTP memory. The Root of Trust Private Key (ROTRK) is property of the Original Equipment

Manufacturer (OEM); and (2) the AP ROM Bootloader (BL1) image stored on the Application Processor (AP) Trusted ROM. At power on, BL1 executes and validates Trusted Boot Firmware (BL2) that does the same with three Third Stage Bootloader (BL3x) images.

The HUK is another important RoT element. The HUK is a per-device unique symmetric key generated by the OEM and written to the SoC's OTP memory [4]. The HUK is accessible only inside the secure world and is used to derive other keys, providing the TEE with seal and unseal primitives [35]. The Unique SoC Identifier (SoC_ID) [3] is calculated from the HUK and the ROTPK with: $\mathit{SoC\_ID} = \mathit{SHA\text{-}256}(\mathit{ROTPK} \parallel \mathit{AES}_{\mathit{HUK}}(\mathit{Fixed\ Pattern}))$

2.2 Cryptographic Schemes

In *Matchmaking Encryption* [5], there is a sender that sends a message to a receiver. The sender and the receiver agree on policies the other must satisfy to reveal the message. A message is revealed when a *match* occurs, i.e., when the sender's attributes satisfy the policies established by the receiver and the receiver's attributes satisfy the policies established by the sender.

Matchmaking Encryption assures that during message decryption no information about the parties' policies is leaked beyond the fact that a match did or did not occur, namely which policy failed when there is a mismatch, therefore preserving sender and receiver anonymity (DG2). Furthermore, Matchmaking Encryption is non-interactive (DG1), meaning that the communicating parties do not need to be online at the same time to authenticate to each other or to exchange data.

In TRX we use a construction of Matchmaking Encryption called Identity-based Matchmaking Encryption. In IB-ME access policies are simply bit-strings that represent identities. Therefore, an attribute $x \in \{0,1\}^*$ only satisfies the access policy $\mathbb{A}$ if $\mathbb{A} = x$, i.e., the sender s and the receiver r just specify a single identity in place of general policies.

In relation to Matchmaking Encryption, the decryption algorithm of IB-ME does not require a decryption key $DK_{\mathbb{S}}$ associated with a policy $\mathbb{S}$ chosen by the receiver and satisfied by the sender's attributes. This not only simplifies this Matchmaking Encryption construction, but also makes it more scalable since a receiver does not have to request different $DK_{\mathbb{S}}$, from the Trusted Authority, for different senders. Notice that this does not ensure the anonymity of the sender to the receiver after the ciphertext has been decrypted, since a successful decryption implies that the receiver specified the sender's identity string; however, this is not a requirement of our anonymity property (see the definition of DG2).

IB-ME relies on a Trusted Authority that yields a Master Public Key (MPK) and a Master Secret Key (MSK) generated by a *Setup* function on input 1^{λ}, where $\lambda \in \mathbb{N}$ is the security parameter: $(\mathit{MPK}, \mathit{MSK}) \leftarrow \mathit{Setup}(1^{\lambda})$. The MPK is published, and both the sender and the receiver have access to it.

The encryption key EK_s, associated with the sender identity s, is generated by the Trusted Authority with an *SKGen* function: $\mathit{EncryptionKey}(\mathit{EK})_s \leftarrow \mathit{SKGen}(\mathit{MSK}, s)$. Afterwards, EK_s is sent to the sender s. In a similar manner, the decryption key DK_r, associated with the receiver identity r, is generated by the Trusted Authority with a *RKGen* function: $\mathit{DecryptionKey}(DK)_r \leftarrow \mathit{RKGen}(\mathit{MPK}, \mathit{MSK}, r)$. The DK_r key is sent to the receiver r, e.g., by the Trusted Authority.

Having the encryption key EK_s, the sender s can encrypt a message m specifying the receiver identity *r'* as an access policy resulting in a ciphertext c: $c \leftarrow \mathit{Enc}_{\text{IB-ME}}(\mathit{MPK}, EK_s, r', m)$. Upon receiving the ciphertext c, the receiver r can try to decrypt it with the decryption key DK_r and specifying the sender identity *s'*: $m' \leftarrow \mathit{Dec}_{\text{IB-ME}}(\mathit{MPK}, DK_r, s', c)$. If a match occurs ($r = r' \wedge s = s'$), the message is revealed ($m' = m$). In case of a mismatch ($r \neq r' \vee s \neq s'$), the decryption function returns an error ($m' = \perp$).

Another cryptographic scheme, *Authenticated Encryption with Associated Data (AEAD)* [27], is used to ensure confidentiality and integrity (DG3, DG4). Specifically, TRX encrypts the Trusted Volume data with AEAD. AEAD has operations $\mathit{Enc}_{AEAD}(K, N, P, A) \rightarrow (C, T)$ for authenticated encryption and $\mathit{Dec}_{AEAD}(K, N, C, A, T) \rightarrow P'$ for authenticated decryption. Enc_{AEAD} has four inputs: a secret key K, a nonce N, plaintext P and Additional Authenticated Data (AAD) A. The output consists of a ciphertext C and a tag T. P is encrypted, producing C, but A is not encrypted. T authenticates P and A. Dec_{AEAD} has five inputs: K, N, C, A and T, as above. If all inputs are authentic, P is revealed ($P' = P$), otherwise an error is returned ($P' = \perp$).

3 TRX: A Data Relocation Service

3.1 TRX Service

TRX is a trusted storage service for TrustZone TEEs that supports transferring TVs with Persistent Objects (POs) to the TrustZone TEEs of other devices. IB-ME is used to share and import the POs with the four properties DG1-DG4. All TRX sensitive operations are performed within the TEE; on the contrary, the REE is entrusted only with confidentiality-and-integrity-protected data. A TUI is assumed to exist for the user to grant/deny authorization.

Figure 1 shows how TRX can be used to *transfer data* (a TV with several POs) between two devices. TRX organizes POs in shareable bundles called TVs (one TV is a set of POs). Each TV is protected with a master key (TVK), then stored in a directory on the REE File System (FS). TRX shares a TV with another device by sharing both the cryptographically protected TV and its TVK. Each device has a Unique Device Identifier (UDID). The sender encrypts the TVK using the $\mathit{Enc}_{\mathit{IB\text{-}ME}}$ function and specifying the receiver UDID. TRX imports a foreign TV by decrypting its TVK. The receiver decrypts the TVK using the $\mathit{Dec}_{\mathit{IB\text{-}ME}}$ function and specifying the sender UDID. The transfer itself is made offline using some storage device, e.g., a flash drive.

TRX also supports the *backup and recovery* of TVs. The process is similar to the one shown in the figure with two changes. First, the receiver device is the same as the sender device. Second, the storage device is not used to transfer

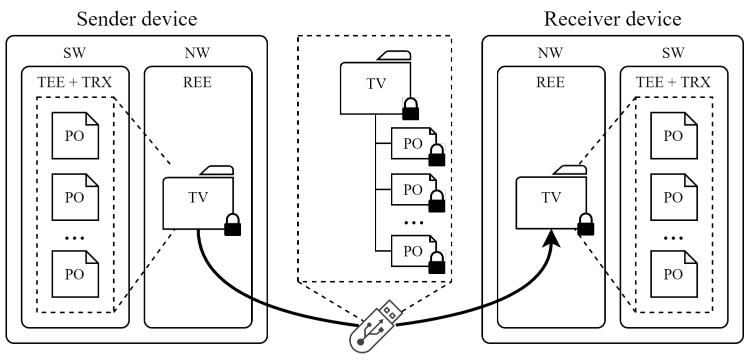

Fig. 1. Sharing a TV with TRX

the TV to the receiver device but to/from the backup device (or it can be the backup device itself).

3.2 IB-ME use in TRX

In Identity-Based Encryption (IBE), entities obtain private keys associated with their identities from a Trusted Authority; the public key is the identity/identifier that has an application-dependent format, e.g., an e-mail address. This process involves no direct communication between potential sender and receiver participants: they only interact once with the Trusted Authority, during setup.

IB-ME, used by TRX, also allows expressing the policies necessary to decrypt the message (TVK) in terms of the identities of the sender and the receiver. With this scheme, the TVK is revealed only if the specified identities match those of the participants; otherwise, nothing is leaked, namely, the identity of the participants, so anonymity (DG2) is satisfied. The data inside a TV is encrypted with AEAD, which provides confidentiality (DG3) and integrity (DG4) [27].

In TRX, the IB-ME keys of each participant are kept protected inside its TEE, and the relocation of data between TEEs is carried out by encrypting a TVK and specifying the receiver identity. A relocated TV is imported by decrypting the TVK. Relocating and importing TVs requires no interaction between the sender and receiver devices, i.e., it can be carried out in a non-interactive way (DG1), e.g., using a flash drive or a dead drop. TRX provides an API that allows applications running in the secure world to write and read persistent data objects, and to share and import TVs.

Before sharing a TV, the system requests user authorization using a TUI (e.g., a touchscreen) such as those that extend TrustZone [7]. A TUI is a user interface controlled by the TEE and isolated from the REE [13] and provides: (1) Secure Display – information displayed to the user cannot be observed, modified or removed by REE software. TZASC and TZPC are used to switch the control of the display to the secure world and protect the display data [21]. (2) Secure Input – information entered by the user cannot be observed or modified by

REE software. The Generic Interrupt Controller (GIC) is used to register TEE interrupt handlers for the input device and isolate the control of the device from the normal world [19]. (3) Secure Indicator (optional) – indication that the displayed screen can be considered trusted by the user. An LED under exclusive control of the TEE can be used to indicate when the screen is controlled by the TEE [34].

3.3 Threat Model and Trust Assumptions

We trust the device's hardware, including the TrustZone extensions, the SoC's OTP memory and trusted ROM. We trust the OEM for securing the ROTRK, the software vendors of the TBB images for securing the secure world private key and the BL3x private keys, and the TEE software vendor for securing the TEE private key (Section 2.1).

The Secure Monitor and the secure world software stack are trusted. The normal world software stack, including the OS, device drivers and libraries, is not trusted. The network is also not trusted. Denial of Service (DoS) attacks issued by the REE OS or any individual with physical access to the device or its peripherals are not considered. Side-channel attacks [20] and other physical attacks that fall outside the defense capabilities of ARM TrustZone are also not considered.

The Trusted Authority is also trusted, similarly to the Attestation Service used in all SGX solutions [18], or a Certificate Authority in a PKI. We assume the assumptions of the cryptographic schemes are held, so they work as expected.

3.4 System Architecture

Figure 2 shows the architecture of a device with TRX, where grayed boxes represent new TRX components or original TrustZone components that were modified for TRX. The original TrustZone system has the following components: (1) *Trusted Kernel*: a program that provides run-time support for TAs: cross-world and cross-TA communication management, TUI management and a Kernel Managed Trusted Storage (KTS) service. The KTS implements a *hybrid setting* in which data is encrypted inside the TEE and then stored in the file system of the REE, and a *master hash* that protects this data is stored in the TEE. The trusted kernel has a driver for controlling the User Interface (UI) device leveraged for TUI; (2) TAs: TEE user-level applications. Each TA has a Unique Trusted Application Identifier (TA_ID). TAs are signed with the TEE software vendor private key and the signature covers the TA_ID. The TEE software vendor public key is embedded into the trusted kernel for authenticating the TAs; (3) Storage device: a storage device with support for rollback-protection, such as an embedded Multi-Media Controller (eMMC) device. This storage device backs the REE FS and has a rollback-protected region, such as a Replay Protected Memory Block (RPMB) partition, that is controlled by the TEE; (4) *Internal API*: an interface that exposes the core services of the trusted kernel to the TAs. The Internal API provides functions for cross-world communication and

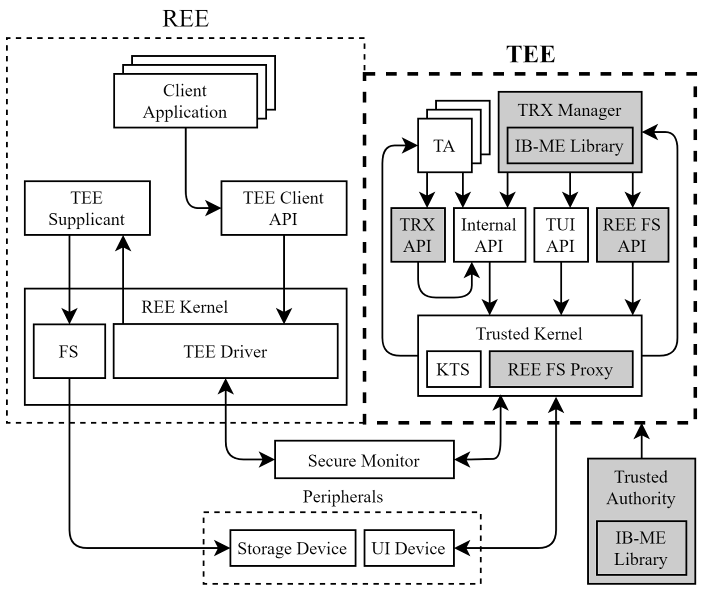

Fig. 2. Architecture of a device with TRX, plus a Trusted Authority

cross-TA communication, and for accessing the KTS; (5) *TUI API*: allows the secure display of information to the user and the secure input of information from the user. The UI wields a *Secure Indicator* such as a LED, under exclusive control of the TEE; (6) *TEE Supplicant*: user-level service running in the REE that listens to requests from the TEE and performs operations on the REE FS on its behalf; (7) *TEE Client API*: allows REE applications to perform requests to TAs; (8) TEE Driver: driver used by the REE kernel to handle low-level interactions with the TEE.

TRX extends the base TrustZone architecture with the following components: (1) *TRX API*: an interface that exposes the TRX *write*, *read*, *share* and *mount* functions to TAs (see Table 1); (2) *TRX Manager*: a TA responsible for managing TRX key materials, handling data encryption and decryption, and enforcing access control of TRX resources and operations; (3) *Trusted Authority*: an entity that issues IB-ME keys; (4) *IB-ME Library*: provides the IB-ME functions: *Setup*, *SKGen*, *RKGen*, $Enc_{IB\text{-}ME}$ and $Dec_{IB\text{-}ME}$. The library also defines the IB-ME key types: *MPK*, *MSK*, *EK* and *DK*. (5) REE FS Proxy: a trusted kernel service which allows TAs to access the REE FS by forwarding FS operation requests to the TEE Supplicant; (6) *REE FS API* exposes REE FS file operation functions to the TAs.

TRX protects the POs in a device with the following types of keys: (1) TVK – a randomly generated per-TV key, used to derive the Trusted Storage Space Key. Each TVK is stored on the TV record of the TV it protects; (2) Trusted Storage Space Key (TSSK) – a key derived from a TVK and a TA_ID. A

Function	**Description**
$write(path, data)$	Write data into PO
$read(path) \rightarrow data$	Read data from PO
$share(UDID, mountpoint, label) \rightarrow (C, dir)$	Share TV
$mount(UDID, mountpoint, C, dir)$	Mount shared TV

Table 1. The Trusted API of TRX

TSSK protects a Trusted Storage Space (TSS) within one TV. The TSSK is not persistently stored as it can be deterministically derived from the TVK; (3) Persistent Object Key (POK) – a randomly generated key for protecting a PO. PO data is encrypted with the POK and stored in the body of the PO file. The POK is encrypted with the TSSK and stored in the header of the PO file.

3.5 Workflow

TAs use the TRX API to perform the TRX operations: *write*, *read*, *share* and *mount*. The TRX Manager has an entry point for each operation. The TRX API uses the Internal API to request the trusted kernel for a *session* with the TRX Manager and to call a TRX Manager's entry point. The trusted kernel establishes the session and provides the TRX Manager with a set of client properties. This set of properties includes the client type, which may be a REE application or a TA, and the TA_ID, if the client is a TA. The TRX Manager filters each request and enforces the following access policies: (1) Only TA clients are granted access to the TRX operations; (2) The *write* and *read* entry points select the client's TSS based on its TA_ID; (3) The *share* and *mount* entry points require user authorization via TUI.

The TRX Manager persistently stores data using the KTS and the REE FS Proxy. The Internal API interacts with the KTS, whereas the REE FS API interacts with the REE FS Proxy. Each function of the REE FS API calls an Supervisor Call (SVC) function implemented in the REE FS Proxy. The REE FS Proxy sends REE file operation commands to the TEE Supplicant through a series of Remote Procedure Call (RPC) requests [29]. The TEE Supplicant performs the requested operations on the REE FS and responds. The TRX Manager encrypts the plaintext data before using the REE FS Proxy to store it to REE FS files. The KTS stores the Trusted Volume Table (TVT) which contains key materials for protecting the REE FS files. The KTS implements the *hybrid setting* (cf. Section 3.4), so a *master hash* of the KTS is stored to the TEE-controlled storage region. When the TRX Manager updates the TVT, the KTS updates the *master hash*, providing integrity (DG4) and rollback-protection to the TVT.

When a TA requests the *share* and *mount* operations to the TRX Manager, the TRX Manager uses the TUI API to request user authorization for the operation. The TUI API calls the SVC functions implemented in the TUI management service of the trusted kernel. The TUI management service uses a device driver to handle low-level interactions with a UI device, such as a touchscreen.

Algorithm 1 TRX Write Operation

```
 1: function write(path, data)
 2:     Get $(M, PO\_ID) \leftarrow path$
 3:     Get UDID
 4:     Read TVT from KTS
 5:     if TVT has record of TV mounted on $M$ then
 6:         Get $(TVK, V_{TSST}, UDID_{TSST}, d) \leftarrow$ TV record
 7:         Read $C_{TSST}$ from $d/f_{TSST}$
 8:         $TSST \leftarrow Dec_{TRX}(TVK, TA\_ID_{MGR}, V_{TSST}, UDID_{TSST}, PO\_ID_{TSST}, C_{TSST})$
 9:     else
10:         Generate $d$ and random TVK
11:         Initialize TSST and set $V_{TSST} \leftarrow 0$
12:         Add TV record $\leftarrow (M, TVK, V_{TSST}, UDID, d)$
13:     end if
14:     Get $TA\_ID_{CLI} \leftarrow$ client properties
15:     if TSST has TSS record for $TA\_ID_{CLI}$ then
16:         Get POT $\leftarrow$ TSS record
17:     else
18:         Initialize POT
19:         Add TSS record $\leftarrow (TA\_ID_{CLI}, POT)$
20:     end if
21:     if POT has PO record for PO_ID then
22:         Get $(V_{PO}, f_{PO}) \leftarrow$ PO record
23:     else
24:         Generate $f_{PO}$ and set $V_{PO} \leftarrow 0$
25:         Add PO record $\leftarrow (PO\_ID, V_{PO}, UDID, f_{PO})$
26:     end if
27:     Increment $V_{PO}$
28:     $C_{PO} \leftarrow Enc_{TRX}(TVK, TA\_ID_{CLI}, V_{PO}, UDID, PO\_ID, data)$
29:     Write $C_{PO}$ to $d/f_{PO}$
30:     Update PO record with $V_{PO}$ and UDID
31:     Increment $V_{TSST}$
32:     $C_{TSST} \leftarrow Enc_{TRX}(TVK, TA\_ID_{MGR}, V_{TSST}, UDID, PO\_ID_{TSST}, TSST)$
33:     Write $C_{TSST}$ to $d/f_{TSST}$
34:     Update TV record with $V_{TSST}$ and UDID
35:     Write TVT to KTS
36: end function
```

The Trusted Authority uses the IB-ME library to generate IB-ME keys. The TRX Manager uses the IB-ME library and the IB-ME keys to encrypt a TVK when a TV is being shared, and to decrypt a TVK when a TV is being mounted.

Write is the TRX operation for inserting data in a PO. Algorithm 1 shows the procedure of the *write* operation, which has two inputs: the PO path and the plaintext data. On lines 2 to 4, the mount point M and the PO Identifier (PO_ID) are parsed from the PO path ($path = M/PO\_ID$), then the device's UDID is fetched and the TVT is read from the KTS using the Internal API. If the TVT has a record of a TV mounted on M (lines 5 to 8), the TVK, the Trusted Storage Space Table (TSST) version V_{TSST}, the UDID of the device which wrote the TSST $UDID_{TSST}$ and the REE directory path d of the TV are read from the TV record. Then, the encrypted TSST C_{TSST} is read from the TSST file of d using the REE FS API. The filename of a TSST file f_{TSST} is a fixed string defined in the TRX Manager (e.g., *tsst.trx*). Then, C_{TSST} is decrypted with Dec_{TRX}. Dec_{TRX} receives the TVK, the TA_ID of the TRX Manager TA_ID_{MGR}, V_{TSST}, $UDID_{TSST}$, the PO_ID of the TSST PO_ID_{TSST} and C_{TSST} as input and outputs the TSST. PO_ID_{TSST} is a fixed string defined in the TRX Manager (e.g., *"TSST"*). If the TVT has no record of a TV mounted on M (lines 9 to 12), a new TV is created and mounted on M. To create a new TV, the TRX Manager randomly generates a TVK, selects a new d, initializes an empty TSST and sets V_{TSST} to zero. Then, the new TV is mounted on M by creating a new TV record on the TVT with M, TVK, V_{TSST}, UDID and d.

Once the TSST of the TV is mounted on M, the TRX Manager uses the client's TA_ID, TA_ID_{CLI}, to find the TSS record of the client. The Internal API is used to fetch the TA_ID_{CLI} from the set of client properties provided by the trusted kernel. If the TSST has a TSS record for the client (lines 15 to 16), the Persistent Object Table (POT) of the client's TSS is read from the TSS record. Otherwise, a TSS for the client is created on the TV mounted on M. To create a TSS for the client, an empty POT is initialized and a new TSS record is created on the TSST with the TA_ID_{CLI} and the new POT. Having the POT of the client's TSS, the TRX Manager queries the record of the PO with the specified PO_ID. If the POT has a record of the PO with the specified PO_ID (lines 21 to 22), the PO version V_{PO} and the filename of the PO file f_{PO} are read from the PO record. Otherwise, a new PO is created on the TSS of the client (lines 23 to 25). To create a new PO, the TRX Manager generates a new f_{PO} and sets V_{PO} to zero. Then, a new PO record containing the PO_ID, V_{PO}, UDID and f_{PO} is added to the POT.

Finally, having V_{PO} and f_{PO}, the TRX Manager encrypts the PO with TRX encryption mechanism and stores it to the REE FS (lines 27 to 29). Before encrypting the PO, V_{PO} is incremented. Then, the PO plaintext data is encrypted with Enc_{TRX}. Enc_{TRX} receives the TVK, the TA_ID_{CLI}, the V_{PO}, the UDID, the PO_ID and the plaintext data as input and outputs the encrypted PO C_{PO}. C_{PO} is written to the PO file d/f_{PO} with the REE FS API. After persistently storing the PO to the REE FS, the TSST is updated and persistently stored as well (lines 30 to 33). First, the PO record is updated with the incremented V_{PO} and the UDID. Note that the TSST contains the POT which in turn contains the PO record. Before encrypting the TSST, V_{TSST} is incremented as well. Then, the TSST is encrypted with Enc_{TRX}. Enc_{TRX} receives the TVK, the TA_ID_{MGR}, the V_{TSST}, the UDID, the PO_ID_{TSST} and the TSST as input and outputs C_{TSST}. C_{TSST} is written to the TSST file of d using the REE FS API. After persistently storing the TSST to the REE FS, the TVT is updated and persistently stored as well (lines 34 to 35). First, the TV record is updated with the incremented V_{TSST} and the UDID. Note that the TVT contains the TV record. Finally, the updated TVT is stored to the KTS using the Internal API.

We omit a similar explanation for the other three operations for lack of space.

4 Evaluation

We implemented the TRX prototype as an extension to OP-TEE [30], a TEE implementation for ARM TrustZone (and Linux as REE) compliant with the GlobalPlatform specifications [14]. We extended OP-TEE with the TRX Manager TA, the TRX API, the REE FS API, the IB-ME library and the REE FS proxy. As Secure Monitor, the Trusted Firmware-A (TF-A) implementation is

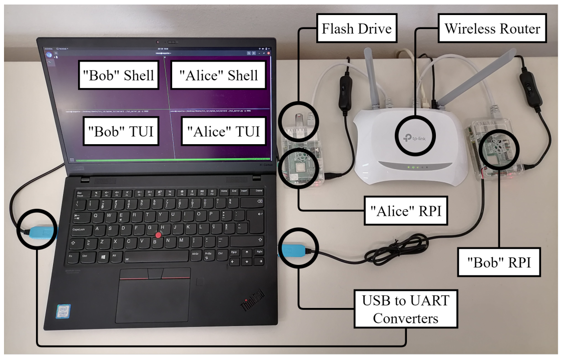

Fig. 3. TRX Prototype

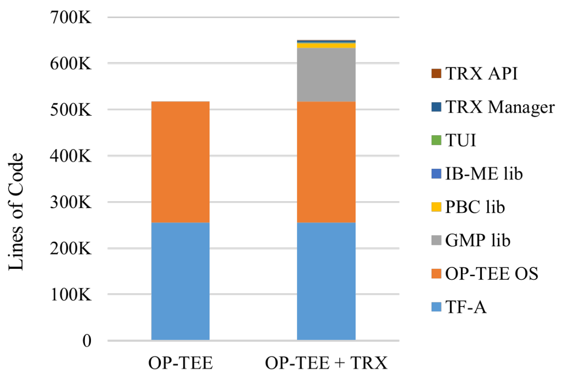

Fig. 4. Impact of TRX on the TCB size

used [31]. TRX was deployed on RPI 3 Model B+ boards. This is a single-board computer with a BroadCom SoC, a quad-core ARM Cortex-A53 with TrustZone, and 1 GB of RAM. The experimental setting is shown in Figure 3.

We evaluated experimentally the impact of TRX on the TCB, the performance of IB-ME library, and the performance of the main operations.

4.1 Impact on the Trusted Computing Base

This section assesses the impact of TRX on the TCB of a TrustZone-enabled system by comparing the TCB size before and after adding our extension to a system with OP-TEE. Figure 4 shows the TCB size of the system with OP-TEE and without/with our implementation of TRX. Before adding TRX, the TCB was composed of TF-A and OP-TEE OS. After adding TRX, the TCB was composed of TF-A, our fork of OP-TEE OS (which includes the REE FS Proxy and the REE FS API), the ports of the GNU Multiple Precision Arithmetic (GMP) and Pairing-Based Cryptography (PBC) libraries, the OP-TEE IB-ME library, the TUI simulator, the TRX Manager and the TRX API. TRX increased the

	Mainstream (Core i5)		OP-TEE (BCM2837B)	
Function	μ (ms)	σ (ms)	μ (ms)	σ (ms)
Setup	2.32	0.55	40.26	0.49
SKGen	3.41	0.96	67.48	0.50
RKGen	4.80	1.57	87.66	0.48
$Enc_{IB\text{-}ME}$	6.38	1.09	173.44	0.67
$Dec_{IB\text{-}ME}$	3.90	0.42	146.18	0.48

Table 2. Execution Time of the IB-ME Functions

TCB size in 25.49%. The GMP library port is the component which increased the TCB the most (+22.38%). This last result was not unexpected as the port of this library was not optimized. A lower impact on the TCB can be achieved by shrinking our port of the GMP library, which was left for future work. Note that originally the PBC library had 65130 lines of code (LoC) and that shrinking reduced the PBC library to 9700 LoC – a reduction of 85.11%. A similar reduction might be achieved with the GMP library, reducing the impact on the TCB size from 22.38% to less than 4%. We estimate that shrinking the GMP library would drop the impact of TRX on the TCB size to less than 8%.

4.2 IB-ME Library Performance

This experiment compares the performance of the IB-ME C library, that we developed for TRX, in two environments: one with an Intel Core i5-8265U CPU and a mainstream Linux system (mainstream version); and another with a Broadcom BCM2837B0 SoC adapted for OP-TEE (OP-TEE version). The Trusted Authority uses the mainstream version and the TRX Manager uses the OP-TEE version.

Table 2 shows the average execution time μ and standard deviation σ of the IB-ME functions for both versions of our IB-ME library. The observed timings suggest that both versions are practical for data encryption and decryption, and for key generation. The OP-TEE version takes, in average, 173.44 ms to encrypt data, thus being responsible for 98.60% of the *share* function average execution time, which is 175.90 ms. To decrypt data, the OP-TEE version takes, in average, 146.18 ms, being responsible for 39.96% of the *mount* function average execution time, which is 368.86 ms. There is a performance discrepancy between the mainstream version and the OP-TEE versions because the Intel CPU is faster. In addition, the OP-TEE version is executed on OP-TEE on a system scheduled by Linux; although the world switch is controlled by the Secure Monitor, the REE determines when the TEE is executed and when it is paused. Therefore, the IB-ME operations of the OP-TEE library take longer to finish.

4.3 Write and Read Operations Performance

This experiment benchmarks the TRX API against the OP-TEE implementation of the GlobalPlatform trusted storage API for Data and Keys to evaluate the relative performance of TRX *write* and *read* operations.

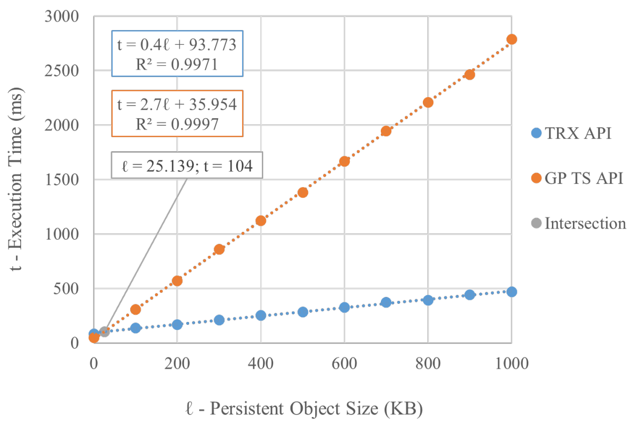

Fig. 5. Write Operation Benchmark

Figure 5 shows the average time t, in milliseconds, required to write a PO of size ℓ, in kilobytes. The performance of writing POs with the TRX API and GlobalPlatform trusted storage API for Data and Keys was modeled with the linear regressions t_{TRX} and t_{OPTEE}, respectively: $t_{TRX} = 0.4\ell + 93.773$ and $t_{OPTEE} = 2.7\ell + 35.954$. The correlation coefficient R^2 of t_{TRX} equals 0.9971 and the correlation coefficient of t_{OPTEE} equals 0.9997. t_{OPTEE} has a bigger gradient than t_{TRX}, meaning that the execution time t required to write a PO of an increasing size ℓ increases at a higher rate with the GlobalPlatform trusted storage API than with the TRX API. The two lines intersect when $\ell = 25.139$, meaning that the GlobalPlatform trusted storage API for Data and Keys is faster than TRX for writing POs smaller than 25 KB and the TRX API is faster for POs larger than 25 KB.

Figure 6 shows the average execution time t, in milliseconds, required to read a PO of size ℓ, in kilobytes. The performance of reading POs with the TRX API and GlobalPlatform trusted storage API for Data and Keys was modeled with the linear regressions t_{TRX} and t_{OPTEE}, respectively: $t_{TRX} = 0.3\ell + 3.5354$ and $t_{OPTEE} = 0.3\ell + 16.175$. The correlation coefficients R^2 of both regressions equals 1 – a perfect positive correlation. Both regressions have the same gradient, meaning that the execution time t required to read a PO of an increasing size ℓ increases at the same rate with both Application Programming Interfaces (APIs). However, the TRX API is, in average, 9 ms faster to read a PO than the GlobalPlatform trusted storage API for Data and Keys.

The results show that TRX outperforms the OP-TEE trusted storage to write POs larger than 25 KB. However, the OP-TEE trusted storage duplicates all fields of its PO system to ensure atomicity while writing a PO and the TRX prototype does not ensure atomicity, which gives TRX a performance advantage over OP-TEE. Implementing a mechanism to ensure the atomicity of the TRX operations is left for future work.

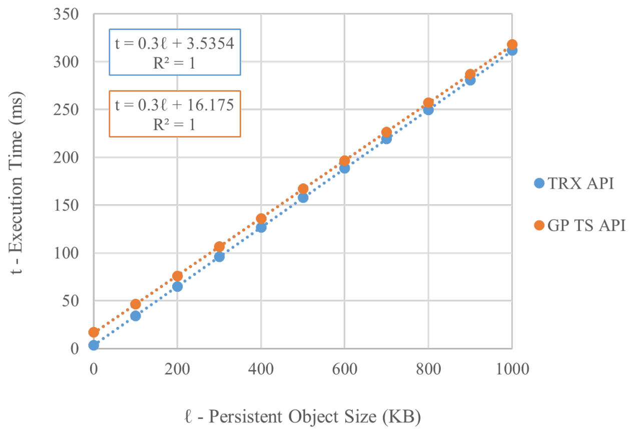

Fig. 6. Read Operation Benchmark

TRX outperforms OP-TEE in reading POs since the OP-TEE trusted storage mechanism reads and decrypts the *dirf.db* file, which lists all files in trusted storage as well as their versions and MAC tags, before reading and decrypting a PO file whereas TRX only needs to read and decrypt the PO file itself due to optimizations in TVT and TSST.

5 Related Work

This section briefly presents the work related to TRX. The section is necessarily brief due to lack of space; many other works exist. Although several considered the use of TrustZone or TEEs for data storage [15, 17, 28], to the best of our knowledge, none focused on the transfer, backup, and recovery of TrustZone-protected data providing properties DG1-DG4.

Intel SGX TEEs, called enclaves, can seal data, i.e., to encrypt enclave data for persistent storage. Sealing keys are derived from a key burned into on-chip OTP memory, thus the sealed data is bound to the device [10]; therefore, Intel Software Guard Extensions (SGX) shares with TrustZone the challenge of accessing sealed data in a different device. Several enclave migration solutions for SGX use a variation of the first alternative presented in the introduction, with its limitations in terms of non-interactivity (DG1) and anonymity (DG2) [1, 22, 25].

There is a large literature on anonymization of communications, but unrelated to TEEs, such as Chaum's Mix-Net that hides the match between sender and receiver by wrapping messages in layers of public-key cryptography, relaying them through mixes [8]. The Tor network does something similar but with better performance and a large practical adoption [12]. Babel is a proposal for anonymity (DG2) in e-mail communications [16].

Matchmaking Encryption is a recent cryptographic scheme [5]. We only found four other works on Matchmaking Encryption: Xu et al. present Matchmaking

Attribute-based Encryption (MABE) and use it to design a secure fine-grained bilateral access control data sharing system for cloud-fog computing [32]; Xu et al. present Lightweight Matchmaking Encryption (LME) and use it in the domain of distributed access control [33]; Lin et al. provide a Functional Encryption for Deterministic Functionalities (FE) [6] construction for Matchmaking Encryption [23]; recently, Certificateless Matchmaking Encryption (CL-ME) was introduced IB-ME [9].

6 Conclusion

We introduce TRX, a trusted storage mechanism that enables TrustZone-TEEs to relocate data to other TEEs with confidentiality, integrity and anonymity guarantees in a non-interactive way. TRX is one of the first systems based on Matchmaking Encryption. We implemented TRX for Raspberry Pi, obtaining a small TCB increase and a delay that is practical.

Acknowledgments This work was supported by the European Commission through contract 870635 (DE4A) and by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UIDB/50021/2020 (INESC-ID).

References

1. F. Alder et al. Migrating SGX enclaves with persistent state. In *2018 48th Annual IEEE/IFIP Intl. Conf. on Dependable Systems and Networks*, pages 195–206, 2018.
2. ARM. ARM security technology: Building a secure system using TrustZone technology, 2009.
3. ARM. Trusted board boot requirements client (TBBR-CLIENT) Armv8-A, 2018. Document number: ARM DEN0006D.
4. ARM. TrustZone for Armv8-A, 2020. Version 1.0.
5. G. Ateniese et al. Match me if you can: Matchmaking encryption and its applications. In *Annual Intl. Cryptology Conf.*, pages 701–731. Springer, 2019.
6. D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In *Theory of Cryptography Conference*, pages 253–273. Springer, 2011.
7. Y. Cai et al. SuiT: Secure user interface based on TrustZone. In *2019 IEEE International Conference on Communications (ICC)*, pages 1–7, 2019.
8. D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.
9. B. Chen et al. CL-ME: efficient certificateless matchmaking encryption for internet of things. *IEEE Internet of Things Journal*, 8(19):15010–15023, 2021.
10. V. Costan and S. Devadas. Intel SGX Explained. *IACR Cryptology ePrint Archive*, 2016(086):1–118, 2016.
11. V. Costan, I. Lebedev, and S. Devadas. Sanctum: Minimal hardware extensions for strong software isolation. In *25th USENIX Security Symp.*, pages 857–874, 2016.
12. R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. *Proc. of the 13th Conf. on USENIX Security Symposium*, 2004.
13. GlobalPlatform, Inc. Trusted User Interface API Version 1.0, June 2013. Document Reference: GPD_SPE_020.

14. GlobalPlatform, Inc. TEE internal core API specification version 1.1.2.50 (target v1.2), June 2018. Document Reference: GPD_SPE_010.
15. J. González and P. Bonnet. TEE-based trusted storage. Technical report, IT University Technical Report Series, 2014.
16. C. Gulcu and G. Tsudik. Mixing E-mail with Babel. In *Proceedings of the Symposium on Network and Distributed Systems Security*, pages 2–16, 1996.
17. S. Huang, C. Liu, and Z. Su. Secure storage model based on TrustZone. In *IOP Conference Series: Materials Science and Engineering*, 2019.
18. Intel. *Attestation Service for Intel Software Guard Extensions: API Documentation.* Intel Corporation, 2020. Revision 6.0.
19. M. Lentz et al. SeCloak: ARM TrustZone-based mobile peripheral control. In *Proc. of the 16th Annual Intl. Conf. on Mobile Systems, Applications, and Services*, 2018.
20. L. Lerman, G. Bontempi, and O. Markowitch. Side channel attack: an approach based on machine learning. *Center for Advanced Security Research Darmstadt*, pages 29–41, 2011.
21. W. Li et al. Building trusted path on untrusted device drivers for mobile devices. In *Proceedings of 5th ACM Asia-Pacific Workshop on Systems*, 2014.
22. H. Liang, Q. Zhang, M. Li, and J. Li. Toward migration of SGX-enabled containers. In *2019 IEEE Symposium on Computers and Communications*, pages 1–6, 2019.
23. X.-J. Lin and L. Sun. Matchmaking encryption from functional encryption for deterministic functionalities. https://www.researchgate.net/, 2020.
24. F. McKeen et al. Innovative instructions and software model for isolated execution. In *Proc. of the 2nd Intl. Workshop on Hardware and Architectural Support for Security and Privacy*, June 2013.
25. J. Park, S. Park, B. B. Kang, and K. Kim. eMotion: An SGX extension for migrating enclaves. *Computers & Security*, 80:173–185, 2019.
26. S. Pinto and N. Santos. Demystifying Arm TrustZone: A Comprehensive Survey. *ACM Computing Surveys*, 51(6):130, 2019.
27. P. Rogaway. Authenticated-encryption with associated-data. In *Proc. of the 9th ACM Conference on Computer and Communications Security*, pages 98–107, 2002.
28. N. Santos, H. Raj, S. Saroiu, and A. Wolman. Using ARM TrustZone to build a trusted language runtime for mobile applications. In *Prof. 19th Int'l Conference on Architectural Support for Programming Languages and Operating Systems*, 2014.
29. R. Thurlow. RPC: Remote Procedure Call Protocol Specification Version 2. RFC 5531, RFC Editor, May 2009.
30. TrustedFirmware. OP-TEE documentation. https://optee.readthedocs.io/, 2021.
31. TrustedFirmware. Trusted Firmware-A Documentation. https://trustedfirmware-a.readthedocs.io/, 2021.
32. S. Xu et al. Match in my way: Fine-grained bilateral access control for secure cloud-fog computing. *IEEE Trans. Dependable Secure Comput. (online first)*, 2020.
33. S. Xu et al. Expressive bilateral access control for internet-of-things in cloud-fog computing. In *Proc. of the 26th ACM Symp. on Access Control Models and Technologies*, pages 143–154, 2021.
34. K. Ying, P. Thavai, and W. Du. Truz-view: Developing TrustZone user interface for mobile os using delegation integration model. In *Proc. 9th ACM Conference on Data and Application Security and Privacy*, pages 1–12, 2019.
35. S. Zhao et al. Providing root of trust for ARM TrustZone using on-chip SRAM. In *Proc. of the 4th Intl. Workshop on Trustworthy Embedded Devices*, 2014.