Practical and Privacy-Preserving TEE Migration

Ghada Arfaoui, Jean-François Lalande, Saïd Gharout, Jacques Traoré

Practical and Privacy-Preserving TEE Migration

Ghada Arfaoui[1,3], Saïd Gharout[2], Jean-François Lalande[3,4], and Jacques Traoré[1]

[1] Orange Labs, F-14000 Caen, France
[2] Orange Labs, F-92130 Issy-Les-Moulineaux, France
[3] INSA Centre Val de Loire, LIFO, F-18022 Bourges, France
[4] Inria, CentraleSupélec, IRISA, F-35576 Cesson-Sévigné, France

Abstract. Trusted Execution Environments (TEE) are becoming widely deployed in new smartphone generation. Running within the TEE, the Trusted Applications (TA) belong to diverse service providers. Each TA manipulates a profile, constituted of secret credentials and user's private data. Normally, a user should be able to transfer his TEE profiles from a TEE to another compliant TEE. However, TEE profile migration implies security and privacy issues in particular for TEE profiles that require explicit agreement of the service provider. In this paper, we first present our perception of the deployment and implementation of a TEE: we organize the TEE into security domains with different roles and privileges. Based on this new model, we build a migration protocol of TEE profiles ensuring its confidentiality and integrity. To this end, we use a reencryption key and an authorization token per couple of devices, per service provider and per transfer. The proposed protocol has been successfully validated by AVISPA, an automated security protocol validation tool.

Keywords: TEE, credential transfer, privacy

1 Introduction

In the last years, a secure mobile operating system running alongside the standard Rich Execution Environment (REE for short), has emerged: the Trusted Execution Environment (TEE for short). A TEE could have its own CPU and memory, and hosts isolated Trusted Applications (TA for short) that provide secure services to the applications running within the REE. These TAs belong to diverse service providers. Each TA manipulates a profile, constituted of secret credentials and user's private data.

The TEE has been standardized by GlobalPlatform, however, to the best of our knowledge, there is no specification or research work that models the TEE internal architecture or ecosystem. For instance, comparing to smart cards, the GlobalPlatform Card Specifications [12] have worked on such a model and it is now widely deployed and accepted by all the stakeholders. This is why we propose to study in which extent we can apply it for the TEE context: we identified the limitations of the GlobalPlatform Card model, in the TEE context, when the user wants to migrate its profile from one TEE to another one.

A user, who has many devices or gets a new one, shall be able to securely migrate his TEE profiles from a TEE to another compliant TEE. This problem of migration is currently poorly addressed by TEE implementations, standardization and only few papers have worked on designing TEE migration protocols [16,19]. Two main solutions can be considered: the straightforward solution, which consists in encrypting the profile (by TEE source), transferring it and decrypting it (by target TEE), or a Trusted Server based solution. These solutions present privacy weaknesses, as discussed in the next sections.

In this paper, we propose a new approach to transfer the TEE profiles from a TEE to another compliant TEE while preserving its privacy. For this purpose, we propose to organize the TEE into security domains (SD) with different roles and privileges.

This paper is organized as follows. In Section 2, we present the TEE technology and describe the problem of profile migration. Then, in Section 3, we describe the previous works and discuss how different are our objectives. We define our assumptions and requirements in Section 4. Then, in Section 5, we give a detailed description of our transfer protocol. Finally, in Section 7, we present the validation of our protocol and we conclude in Section 8.

2 Backgrounds and problem statement

A Trusted Execution Environment (TEE) is completely separated from the Rich Execution Environment (REE). It offers a way to isolate Trusted Applications (TAs) and provide secure functionalities such as cryptographic operations or secure PIN input. As defined by GlobalPlatform [14], three main TEE software components are involved: the trusted OS, the TAs and the hardware secure resources, e.g., trusted peripherals. The TAs can access the trusted resources and exchange with Secure Elements using a private API. From the REE, mobile applications can interact with TAs using public APIs. Additional details about TEEs can be found in [3, 5].

Before using a service of the TEE, which is provided by a service provider, several steps should occur, as shown in Figure 1. (1) User enrollment: the user registers to the service provided by the SP, using a secure channel. This step allows to associate the user identity to a dedicated TA inside the device. (2) The TA is personalized inside the TEE by the SP. The necessary application in the REE is also installed. After this step, the service is active. (3) The user could acquire a new device and wish to *securely* transfer its TEE profiles from the first device to the new one. (4) The user may want to destroy its profile, also defined as *disabling* credentials [17].

In this article we focus on step 3, the migration of a TEE profile. Like step 1, step 3 can be threatened by an external attacker. If we suppose that an attacker may have compromised the rich OS or control the network connection of the smartphone, then the enrollment or migration steps become challenging tasks. Indeed, as shown in Figure 1, as the interactions with a TEE crosses the REE, the attacker may succeed to migrate the user's profile from a victim to the

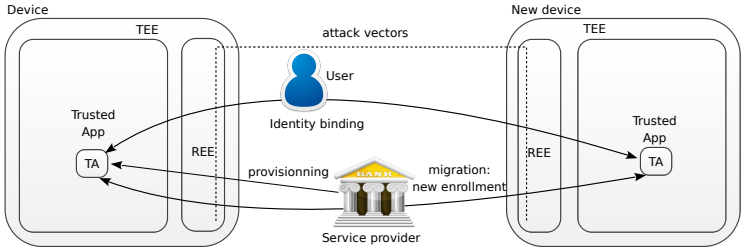


Fig. 1: The life cycle of a TEE service

attacker's smartphone. This attack may succeed because the service provider has no insurance about the TEE security and the user-to-TEE binding. In the next section, we describe the solutions already proposed in the literature in order to show their limitations and motivate a new way of migrating TEE profiles.

3 Related work

The first papers that studied the security of TEE credentials tried to guarantee its local (within the TEE) confidentiality and integrity. For instance in [2], authors proposed to protect TEE data using a unique PUF (Physical Unclonable Functions) AES encryption key for each device. In [18], authors proposed a TEE key attestation protocol proving that a TEE key has been generated inside the TEE and never left this TEE.

Later, scientists have been interested in the enrollment problem (mainly user-device binding) while assuming that there is no operator responsible for the management of the TEE. For instance, Marforio et al. [19] explained that the user's identity can be bound to the device using a password or a SMS or by collecting the device's IMEI. Unfortunately, an attacker that controls the Rich OS can intercept the protocol exchanges. Thus, Marforio et al. proposed some hardware and software modifications in order to secure the enrollment process. Others, like in [16], assumed that the smartphone is *safe* at the first use. This would enable to store a secret password in the TEE.

The problem of credential migration first appears for Trusted Platform Module (TPM), which is in some way the TEE ancestor. The commands of key migration have been specified in TPM specifications [21] and have undergone many improvements. Later, Sadeghi et al. [20] proposed a property based TPM virtualization in order to have a solution that supports software update and migration. The shortcoming of this solution consists in omitting the service provider during the virtual TPM migration. However, some credential migration requires service provider fresh and explicit agreement.

In the specific context of TEE, Kari et al. have proposed a credential migration protocol in open credential platforms [16]. They proposed to make the credential migration user-friendly based on delegated-automatic re-provisioning.

The credentials are backup in clear in a trusted server. Then, the migration process is a re-provisioning from the backup, protected by a secret password only known by the user. The main weakness of this solution lies in the fact that its security, including to user's privacy, is entirely based on a the trustworthy of the trusted server (TS). This latter, as third party, has full access to TEE credentials and user's private data while it is not its owner or provisioner. This proposal implies privacy issues that we propose to solve with our protocol.

GlobalPlatform specifications related to smartcards have been interested in credential management in secure elements (smart cards). However, it seems that the credential privacy in some cases has been overlooked. In GlobalPlatform card specifications [12], the smart card is organized into fully isolated areas called Security Domains (SD). There are a root security domain called ISD for Issuer Security Domain and many Supplementary Security Domains (SSDs) for the different Service Providers. Let us call SPSD the security domain for a given SP. For instance, the ISD could be owned by the Mobile Network Operator (MNO) and the SPSD could be owned by a bank. Once, the SPSD created, there are two modes to manage the content of this SPSD: either directly from SP to SPSD, or through ISD. In the first case, the credential migration process would be naturally implemented in the application of the SP within the smart card: encryption with the target public key, transfer and decryption, provided that the MNO initiates the SP space in the target smart card. In the second case, the MNO plays the role of firewall and proxy for the SPSD without having access to the content between SP and its SSD (SPSD). SPSDs do not have any code enabling to perform a credential transfer.

If we adopt the first model for the TEE, the TEE profile migration would be processed like in the smart card: the TEE profile migration process would be naturally implemented in the TA of the SP: encryption with the target public TEE key, transfer and decryption, provided that the TEE admin - MNO initiates the SP space in the target TEE. As a consequence, each service provider would have to implement a migration process for its service.

If we adopt the second model for the TEE, TEE admin will serve as the single entry point to transfer point-to-point credentials: implementing the migration process would imply privacy issues similarly to the Kari et al. [16] solution. TEE admin would have full access to the SP credentials and user's data in order to encrypt and transfer it. In this paper, we propose a new migration protocol, while adopting the second model, where the TEE Admin plays the role of proxy without having access to SP credentials. We consider the following properties:

- As trusted application profile contains credentials and also personal data (statistics, usage data of the service), during the migration, the profile shall be accessible only by legitimate entities: the SP and the user;
- A special third party, the TEE admin should be responsible of the role of installation, deletion and migration of trusted profiles;
- The trusted application of the SP should not contain any code dedicated to the migration protocol. All the migration software components should be handled by the TEE admin.

4 Attacker model and requirements

We assume that the enrollment, provisioning and personalization processes are already achieved: the trusted application is provisioned to the TEE and has access to its credentials and the user's personal data. By the profile, we mean the credentials (allowing the access to the service) and private data (related to personalization and the use of application/service).

We consider three different actors: the user (the devices' owner), the Service Provider (e.g. the bank) and a TEE admin (e.g., Mobile Network Operator or smartphone manufacturer). We consider the following attacker model:

A1: Communication control. We consider that we have a Dolev-Yao [11] attacker model: an attacker has full control over communication channels.

A2: TEE control. We consider that TEEs are enough resistant to physical attacks according the Protection Profile proposed by GlobalPlatform [13] which is EAL2+ certified.

A3: REE control. Given the possible vulnerabilities of the rich OS, we assume that an attacker can compromise the REE of a user's device.

A4: Entities control and trustworthy. We assume that (i) an attacker cannot spoof the SP, cannot compromise its dedicated spaces within the TEE and the SP is honest, (ii) an attacker cannot spoof the TEE Admin and cannot compromise its dedicated spaces within the TEE, however the TEE Admin can be honest-but-curious and, (iii) the user is honest.

While keeping in view the above discussions, we define the security requirements that a migration protocol shall meet as follow:

R1: Integrity. During the migration process, the integrity of the TEE profile should be ensured. For a given profile, only the user and the relevant service provider should be able to eventually modify the profile content.

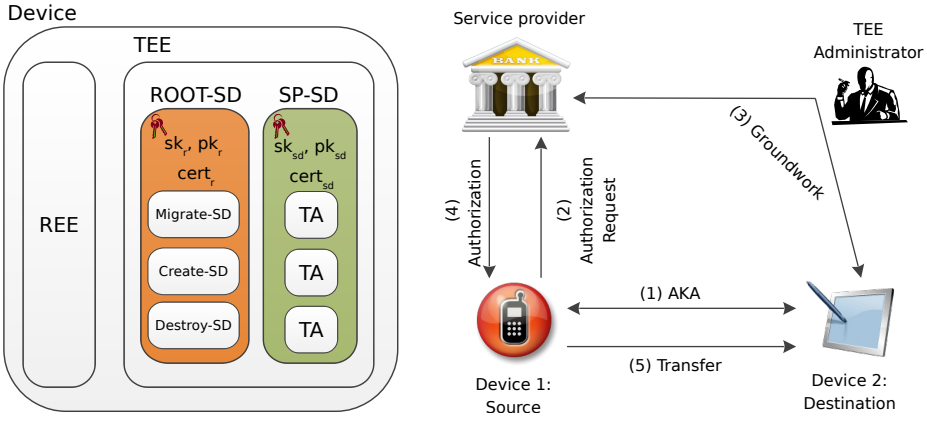
R2: Confidentiality. During the migration process, the confidentiality of the TEE profile should be ensured. For a given profile, only the user and the relevant service provider should be able to eventually read the profile content.

5 TEE migration protocol

Considering the previous requirements, attacker model, and the GlobalPlatform Card Specifications [12], we introduce a new approach for deploying services in a TEE where: TAs of a service provider are hosted in a Security Domain (SD) and a new actor, called TEE admin, has a special SD and implements the migration functionalities. With such a new software architecture, we build a protocol that allows to securely transfer a TEE profile from one device to another one.

5.1 Architecture overview

We organize the TEE into Security Domains (SD) [12]. Every SD can contain one or many Trusted Applications (TA) from the same Service Provider. A SD



(a) Device architecture (b) Protocol overview

Fig. 2: Architecture and protocol overview

is a fully isolated zone. This functionality could be ensured by memory protection mechanisms, firewall functionalities, data isolation techniques implemented at OS level of the TEE, such as the ones used for Linux systems (SELinux, AppArmor,...). For example, in the commercialized TEE solution of Trustsonic, such an architecture can be implemented using containers. A SD manipulates cryptographic keys which are completely separated from any other SD. These keys enable code execution integrity, credentials and user's private data integrity and confidentiality when using a service. Consequently, a SD must not cipher his credentials or user's private data using any external keys whatever is the case, e.g., transfer. We define two types of SD, represented in Figure 2a: (1) SD without management rights (many per TEE): SP-SD (in green). They contain the trusted applications of a service provider. (2) SD with management rights (only one per TEE): ROOT-SD (in orange). It is responsible of creating and deleting SDs, downloading and installing packages in SDs, and also migrating SDs from a TEE to another compliant TEE.

5.2 Protocol overview

In order to migrate a TEE profile from a source device to a target one, the user gets the two devices nearby each other in order to establish a wireless communication, such as NFC or bluetooth. Then, the user starts the migration application, noted Migrate-SD in Figure 2a, within the ROOT-SD of the TEE source. In order to do this, the user must be authenticated in both source and target TEEs. Owing to the authentication procedure, the TEEs check that only the user enrolled by the TEE admin can start the migration process. This authentication can be done through the "Trusted User Interface" [15], or by using the password or the pin code setup at the enrollment phase, or by using a biom-

etry peripheral if available. The next steps of the protocol that involve the two TEEs are presented in Figure 2b and described in the following.

Step 1. An authenticated key agreement takes place between the ROOT-SD of TEE source and the ROOT-SD of target TEE. This prevents the TEE source from disclosing critical information to a malicious environment and prevents the target TEE from accepting malicious data.

Step 2. The TEE source requires a migration authorization from all service providers that have a SD within the TEE source. If a service provider does not agree with the migration of his relevant SD, the migration cannot take place. The migration authorization is temporary and unique per pair of devices, per transfer and per service provider. Indeed, the authorization is related to the date and time of the request that has been initiated. Thus, it is valid only for a given period of time.

Step 3. At that time starts the groundwork for the authorization. First, the service provider checks with the TEE admin whether the target TEE is stated compromised. Then, the service provider checks that the target TEE is not already a client containing a service provider SD. Finally, the service provider asks the TEE admin to set up a specific SD within target TEE, and updates the SD credentials in order to be the unique master of this SD.

Step 4. Finally, the service provider replies to the TEE source with the authorization and necessary migration credentials. The authorization consists of a service provider signature proving his agreement regarding the migration of his SD. The credentials consist of a re-encryption key [8,9]. Using this re-encryption key, the Migrate-SD application will be able to perform the transfer without having access to SD profile. In order to send source profile to the target SD, the source SD provides its profile, encrypted with its public key, to the TA, Migrate-SD, that should re-encrypt it with the re-encryption key. In such a case, even if the TEE Admin is honest-but-curious, it cannot eavesdrop the SD profile.

Step 5. The target TEE must check the validity of the received authorization. At this time, the migration can start.

5.3 TEE profile migration protocol

In the following, we introduce the notations and cryptographic keys used in our protocol. Later, we detail the phases of our protocol: Authenticated key agreement, Service Provider authorization and TEE Profile migration.

Cryptographic keys and notations. We denote (sk_{src}, pk_{src}, $cert_{src}$) (resp. (sk_{tgt}, pk_{tgt}, $cert_{tgt}$)) the TEE private root key and the certified TEE public root key of the source (resp. target) ROOT-SD. These keys are used to authenticate the TEE and set up a key session with an authenticated TEE. A TEE admin is characterized by a private and public key pair (sk_{Admin}, pk_{Admin}). It controls the ROOT-SD and certifies TEE root keys. A Security Domain SP-SD is characterized by a root keys set (sk_{sd}, pk_{sd}, $cert_{sd}$). This is an encryption keys set that enables to securely store SD profile. We denote $SP - SD_{src}$ (resp. $SP - SD_{tgt}$) the service provider SD within TEE source (resp. target TEE).

Enc(pk, M)	: The encryption of the message M using the public key pk.
MAC(sk, M)	: The Message Authenticated Code (MAC) of the message M using the key sk.
$Signature_A$	: The signature on the message sent with the signature using the private key of A.
Verify(pk, σ)	: The verification of the signature σ using the public key pk corresponding to private key sk used during the signature generation.

Fig. 3: Cryptographic notations

Consequently, the tuple ($sk_{SP-SD_{src}}$, $pk_{SP-SD_{src}}$, $cert_{SP-SD_{src}}$) (resp. the tuple ($sk_{SP-SD_{tgt}}$, $pk_{SP-SD_{tgt}}$, $cert_{SP-SD_{tgt}}$)) is the root keys set of $SP-SD_{src}$ (resp. $SP-SD_{tgt}$). A service provider is characterized by (sk_{sp}, pk_{sp}) and a unique identifier ID_{SP}. It provides the security domains root keys and is responsible of the re-encryption key and transfer authorization generation. The notations for the different cryptographic primitives are defined in Figure 3.

Authenticated Key Agreement. The authenticated key agreement (AKA, step 1 in Figure 2b) occurs in order to establish a secure session between two TEEs after a mutual authentication. The AKA can be a password based key agreement [1] or a public key authenticated Key agreement [10] and must be a two ways authentication. In the first case, we can use the password or PIN or biometric data introduced by the user during the authentication phase and in the second case, we can use the TEEs root keys. At the end of this phase, the source and target TEEs will share a couple of ephemeral keys (eK_t, eK_m) to secure the migration. eK_t is the private session key, whereas eK_m is used for MAC computations.

Service Provider Authorization. The TEE source requires a migration authorization from all service providers having trusted applications within the TEE source (step 2 in Figure 2b). This protocol is described in Figure 4. For the sake of simplicity, we consider only one service provider.

(1) The migration application within $ROOT-SD_{src}$ sends the request $INIT\_RQ$ with its signature noted $Signature_{ROOT-SD_{src}}$ to the service provider through the TEE admin. The request includes the identity of the service provider (ID_{SP}), the public key of $SP-SD_{src}$ ($pk_{SP-SD_{src}}$) and the certified TEE public root keys of source and target TEE ($cert_{src}, cert_{tgt}$). (2) When receiving the request, TEE admin checks the certificates ($cert_{src}, cert_{tgt}$), the signature and freshness of the request and a timestamp ($Signature_{ROOT-SD_{src}}$)[5]. It should also check whether source or target TEE are compromised[6] for example using the remote attestation protocols of Baiardi et al. [6]. (3) If checks are successful, the TEE admin transmits the request ($INIT$) to the relevant service provider based on the ID_{SP} received within the request.

(4) With the received request, the service provider checks if the TEE source (resp. target TEE) has (resp. has not) an associated SP-SD by checking if $cert_{src}$ (resp. $cert_{tgt}$) is registered in its database. Then, (5) the service provider inquires TEE admin to create a SP-SD within the recipient TEE via the $SD-Create\_RQ(cert_{tgt})$ command. (6) The TEE admin signs the command (the

[5] TEE implementations like TrustZone offer access to trusted clocks for this usage.

[6] This is already the case for SIM card where MNOs checks if a SIM has been revoked.

signature $Signature_{TEE_{Admin}}$ is performed on the command and a timestamp) and forwards it to the trusted application $Create - SD_{tgt}$ in order to create $SP - SD_{tgt}$. (7, 8, 9, 10). The creation is acknowledged by Ack and $Param$ that are returned to the service provider (through the TEE Admin) in order to let him be able to personalize $SP - SD_{tgt}$, as done classically when personalizing TEE security domains. (11) Once the $SP - SD_{tgt}$ installed, the service provider proceeds to the update of $SP - SD_{tgt}$ credentials in order to have the exclusive control over $SP - SD_{tgt}$ [12].

Finally, the service provider generates the migration authorization. It consists of a re-encryption key K_{proxy} and a signature $PERM$. The signature $PERM$ is computed on the SP identifier ID_{SP}, source and target TEE public keys as well as a timestamp: $PERM = \{ID_{SP}, cert_{src}, cert_{tgt}, TimeStamp\}_{sk_{SP}}$. The re-encryption key K_{proxy} is used to re-encrypt the $SD - SP_{src}$ content such that the result will be understandable only by $SP - SD_{tgt}$: $K_{proxy} = rekeygen(pk_{SP-SD_{tgt}}, sk_{SP-SD_{src}})$. In the literature [8, 9], K_{proxy} is called a proxy key. (12, 13) The authorization is sent to the TEE Admin who signs it and transmits it (with its signature) to $ROOT - SD_{src}$.

TEE Profile Migration. Using the re-encryption key, the confidentiality and integrity of the migration phase is guaranteed. Any outsider cannot eavesdrop the $SP - SD_{src}$ profile and a honest-but-curious TEE Admin has no visibility about the $SP - SD_{src}$ profile. The migration occurs as follows.

As described in Figure 5, $Migrate - SD_{src}$ re-encrypts the protected profile of $SP - SD_{src}$ (P) using the proxy key K_{proxy} to obtain the cipher A. Only $SP - SD_{tgt}$ is able to decrypt A. Afterwards, $Migrate - SD_{src}$ computes B and C. B is the encryption of A and the identifier of the service provider owning $SP - SD_{src}$ (ID_{SP}) using the transfer key eK_t. Regarding C, it is the MAC of A and ID_{SP} using eK_m. At the end of these computations, $Migrate - SD_{src}$ sends A, B, C and $PERM$ to $Migrate - SD_{tgt}$. that proceeds to the verifications of $PERM$ and C. The verification of $PERM$ corresponds to the verification of a signature, its freshness and that its parameters contain the right $cert_{src}$ and $cert_{tgt}$. Next, $Migrate - SD_{tgt}$ decrypts B in order to retrieve A and ID_{SP}. Based on the retrieved ID_{SP}, $Migrate - SD_{tgt}$ transmits A to $SP - SD_{tgt}$.

When the migration finishes, we have two possibilities. On the one hand, the security policy of the service admits to conserve the TEE profile in the source. In such a case, $Migrate - SD_{tgt}$ simply acknowledges that the TEE profile migration is completed successfully (*Signed Ack*). On the other hand, the security policy of the service admits to conserve only one profile. The TEE profile in the source should be destroyed. In order to ensure a fair exchange, exchanges between $Migrate - SD_{src}$ and $Migrate - SD_{tgt}$ must be performed via the service provider. $Migrate - SD_{tgt}$ acknowledges that the TEE profile migration is completed successfully (*Signed Ack*). At this time, $Migrate - SD_{src}$ asks the trusted application $Destroy - SD_{src}$ to destroy the SD corresponding to ID_{SP}. When the operation finishes, $Migrate - SD_{src}$ informs the service provider that the transfer is accomplished. Hence, the service provider will not consider any more TEE source as a client and revoke its corresponding keys.

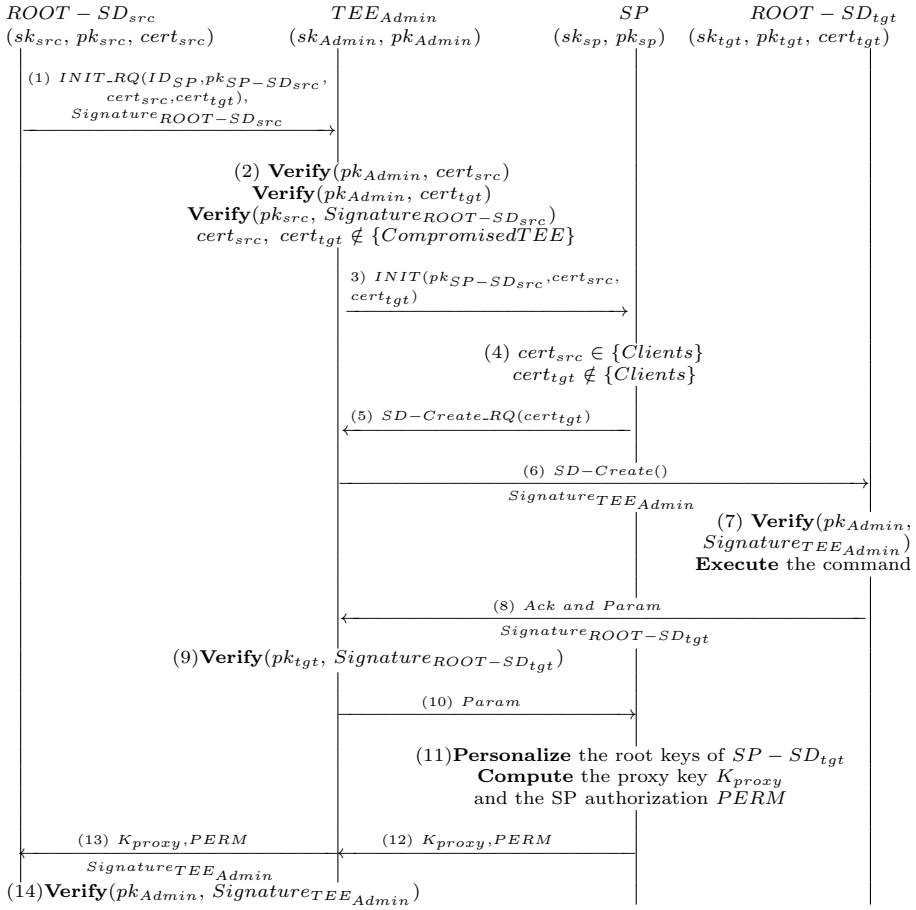


Fig. 4: Service Provider authorization protocol

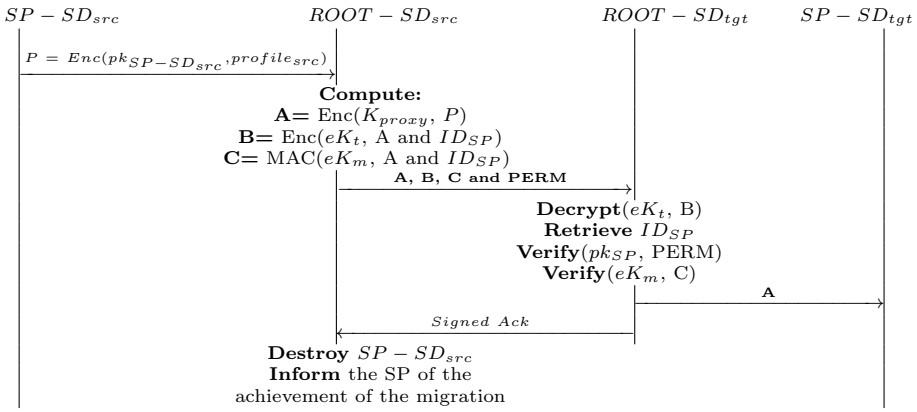


Fig. 5: Profile migration protocol

5.4 Performance remarks

As current TEE implementation does not give access to low level cryptographic primitives we cannot implement the whole protocol. To give an idea of performances, the reader should note that TEEs exploit the CPU of the smartphone with an amount of RAM of some MBs. Thus performance are comparable with what can be obtained in the Rich OS. For example, a RSA computation is achieved in 20 ms on a Galaxy SIII smartphone. Our reencryption scheme needs lower than a RSA computation: we measured, on a Galaxy SIII a reencryption time of about 4 ms.

6 Security analysis

User identification. During a TEE profile migration, it is important to ensure that the target TEE (target device) belongs to the owner of the source TEE (source device). In our model, this is guaranteed by the concept of demonstrative identification [7]. Indeed, we proposed to run the migration protocol over a wireless proximity technology (NFC).

Requirements analysis. During the migration, an outsider or a curious TEE Admin must not be able to read or modify the transferred TEE profile (R1, R2). This is ensured by using the cryptographic re-encryption method. Indeed, the migration authorization, delivered by the service provider, consists of two components: K_{proxy} and $PERM$. $PERM$ is a signature computed by the service provider on $(ID_{SP}, cert_{src}, cert_{tgt}, timeStamp)$. An attacker would not be able to replay this authorization because of the timestamp. Moreover, the transfer process would fail if $cert_{src}$ (resp. $cert_{tgt}$) does not correspond to the certified root public key used by source TEE (resp. target) during the AKA phase. Regarding the re-encryption key K_{proxy}, it is computed based on the private key of the source SD and the public key of the target SD. This means that a cipher text of source SD, if encrypted using K_{proxy}, will be converted to a cipher text of target SD. Thus, only source SD, target SD and the service provider have access (read / modify) to the TEE profile. If the re-encryption key K_{proxy} is improperly computed, the attacker cannot get the TEE profile content.

TEE Admin trustworthy. Besides the cryptographic solution, our approach relies on the trustworthy of the TEE Admin. We assume that a TEE Admin can only be honest-but-curious and not malicious (compromised). Indeed, a malicious TEE Admin can get access to service provider credentials and user's private data. However, we estimate that the assumption of a honest-but-curious TEE Admin is reasonable. This is because a malicious TEE Admin (when detected) risks not only huge financial damages but also his reputation. Knowing that this role should be played by the device manufacturer or the mobile network operator. In our opinion, this risk is far from being taken.

7 Protocol validation

We validated our protocol using the AVISPA [4] tool web interface. AVISPA is an automated tool for the validation of security protocols. It takes as input a protocol modelled in High-Level Protocol Specification Language (HLPSL). This latter is translated into Intermediate Format (IF) and forwarded to the back-end analyser tools.

In Appendix A, we show (the core subset of) our migration protocol model written in HLPSL. In this validation model, we mainly focused on Service Provider authorization protocol (Figure 4) and Profile transfer protocol (Figure 5). Therefore, we assumed that the user authentication and AKA steps have been successfully achieved. Moreover, for the sake of simplicity, we did not consider the $SP - SD_{tgt}$ root key personalization.

We modeled our transfer protocol into six roles in addition to two standard roles (i.e. "`session`" and "`environment`"). First, the role "`sdSrc`" (resp. "`sdTgt`") refers to the $SP - SD_{src}$ (resp. $SP - SD_{tgt}$). Then, the role "`src`" (resp. "`tgt`") represents the Migration TA within TEE source (resp. target TEE). At last, "`teeAdmin`" and "`sp`" respectively correspond to the entities TEE admin and Service Provider. Every role is modelled into a state transition system. A state represents the reception and/or the transmission of a message from our protocol. For instance, "`State` $= 0$" in the role "`teeAdmin`" corresponds to the reception of $INIT\_RQ$ by the TEE administrator in Figure 4. Regarding the role called "`session`", it represents a single session of our protocol where all the other roles are instantiated.

All the roles communicate over Dolev-Yao [11] channels (`channel(dy)`), i.e., an adversary can fully control the communication channels (A1). The attacker knowledge is defined by the set of constants or variables of the `intruder_knowledge` set in the main role (`environment`) (A2, A3). Then, the intruder actions are modeled by the combination of several sessions where the intruder may take part of the sessions running. On the subject of our protocol, besides the initialization of `intruder_knowledge`, we modeled our attacker by the variable `i` (`i` for intruder) such that he can play the role of a honest-but-curious TEE Admin (Line 125) or a honest-but-curious Migration TA in the source (Line 123) or target TEEs (Line 124) (A4-ii). Finally, we note that the attacker i did not compromise the SP nor its SDs (A4-i) because the roles "sdsrc", "sdtgt" and "spagent" are not played by the attacker in the initialized sessions (Lines 122, 123, 124, 125).

The migration authorization, delivered by the service provider, consists of two components: K_{proxy} and $PERM$. $PERM$ is a signature computed by the SP on $(ID_{SP}, cert_{src}, cert_{tgt}, timeStamp)$. Regarding K_{proxy}, it is not a standard cryptographic tool. Thus, AVISPA does not have its predefined predicate. Our model must manually put up all its features. We designed the proxy re-encryption concept owing to the predicate $\wedge$`equal({EncSD}_KProxy,{SDCred}_PKSDtgt)` at the end of the role "`sdSrc`". This predicate models the equality between "the encryption of `EncSD` (the encryption of `SDCred` using the public key of the source SD) using the proxy key" and "the encryption of `SDCred` using the public key of the target

SD". If this equality does not hold, it means that K_{proxy} is a fake key from an attacker which should be assimilated to a denied authorization of the SP.

The HLPSL language provides four predicates to model security requirements. The predicate `secret(E, id, S)` declares the information `E` as secret shared by the agents of set `S`. This security goal will be identified by `id` in the goal section. In addition, `witness`, `request` and `wreuqest` are used to model authentication goals. Regarding the security requirements R1 (integrity) and R2 (confidentiality with respect to outsiders and a curious TEE Admin), we defined them in one goal owing to the predicate `secret(SDCred, transfer, {SDsrc, SP, SDtgt})` at the end of the role "`sdSrc`". This predicate expresses that the content of an SD should remain secret between the SD of TEE source, the service provider and the SD of the target TEE.

We successfully validated our protocol with two AVISPA back-ends (AtSe and SATMC). The AtSe back-end extracts attacks that defeat the security properties by translating the model in constraints on the adversary's knowledge. Using a unification algorithm it integrates at each step of the protocol the new constraints. As our protocol is loop free, the search of possible attacks is complete. Regarding the SATMC back-end, it translates the protocol in propositional formulas that can feed an off-the-shelf SAT solver.

8 Conclusion

In this paper, we have introduced a TEE architecture based on security domains. The root security domain is controlled by the TEE admin and the other security domains isolate the service providers trusted applications. With such an architecture, we have proposed a practical and privacy-preserving TEE profile migration protocol. This protocol requires the dynamic interaction of the service provider and the TEE admin. Owing to the security and functional characteristics of the used re-encryption method, the integrity and the confidentiality of the TEE profile, with respect to external attackers and TEE Admin, are guaranteed. Finally, we successfully validated our protocol using the AVISPA tool.

References

1. Supplemental access control (PACEv2): Security analysis of PACE integrated mapping. In D. Naccache, editor, *Cryptography and Security: From Theory to Applications*, volume 6805 of *LNCS*, 2012.
2. M. Areno and J. Plusquellic. Securing trusted execution environments with PUF generated secret keys. In *11th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 1188–1193, Liverpool, England, UK, June 2012. IEEE Computer Society.
3. G. Arfaoui, S. Gharout, and J. Traoré. Trusted execution environments: A look under the hood. In *2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 259–266, Oxford, UK, April 2014. IEEE Computer Society.

4. A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Drielsma, P. Heam, O. Kouchnarenko, J. Mantovani, S. Modersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Vigano, and L. Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In K. Etessami and S. Rajamani, editors, *17th International Conference on Computer Aided Verification*, volume 3576 of *LNCS*, pages 281–285. Springer, Edinburgh, Scotland, UK, 2005. http://www.avispa-project.org.
5. N. Asokan, J. E. Ekberg, and K. Kostiainen. The untapped potential of trusted execution environments on mobile devices. *IEEE Security And Privacy*, 12(4):293–294, Aug. 2013.
6. F. Baiardi, D. Cilea, D. Sgandurra, and F. Ceccarelli. Measuring semantic integrity for remote attestation. In L. Chen, C. Mitchell, and A. Martin, editors, *2nd International Conference on Trusted Computing*, volume 5471 of *LNCS*, pages 81–100. Springer, Oxford, UK, 2009.
7. D. Balfanz, D. K. Smetters, P. Stewart, and H. C. Wong. Talking to strangers: Authentication in ad-hoc wireless networks. In *Network and Distributed System Security Symposium*, San Diego, California, USA, 2002. The Internet Society.
8. M. Blaze, G. Bleumer, and M. Strauss. Divertible protocols and atomic proxy cryptography. In *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 127–144, Helsinki, Finland, may 1998. Springer.
9. S. Canard, J. Devigne, and F. Laguillaumie. Improving the security of an efficient unidirectional proxy re-encryption scheme. *Journal of Internet Services and Information Security (JISIS)*, 1(2/3):140–160, Aug. 2011.
10. J.-S. Coron, A. Gouget, P. Paillier, and K. Villegas. SPAKE: A single-party public-key authenticated key exchange protocol for contact-less applications. In R. Sion, R. Curtmola, S. Dietrich, A. Kiayias, J. Miret, K. Sako, and F. Seb, editors, *Financial Cryptography and Data Security*, volume 6054 of *LNCS*, pages 107–122, Tenerife, Canary Islands, Spain, 2010. Springer.
11. D. Dolev and A. C. Yao. On the security of public key protocols. In *22Nd Annual Symposium on Foundations of Computer Science*, SFCS '81, pages 350–357, Nashville, USA, 1981. IEEE Computer Society.
12. GlobalPlatform Card technology. Card specification - v2.2.1, Jan. 2011.
13. GlobalPlatform Device Committee. TEE protection profile version 1.2, public release, gpd_spe_021, November 2014.
14. GlobalPlatform Device technology. TEE system architecture, v1.0, Dec. 2011.
15. GlobalPlatform Device technology. Trusted user interface API, v1.0, June 2013.
16. K. Kostiainen, N. Asokan, and A. Afanasyeva. Towards user-friendly credential transfer on open credential platforms. In J. Lopez and G. Tsudik, editors, *9th International Conference on Applied Cryptography and Network Security*, volume 6715 of *LNCS*, pages 395–412. Springer, Nerja, Spain, june 2011.
17. K. Kostiainen, N. Asokan, and J.-E. Ekberg. Credential disabling from trusted execution environments. In *15th Nordic Conference on Secure IT Systems*, number 2, pages 171–186, Espoo, Finland, Oct. 2012. Springer.
18. K. Kostiainen, A. Dmitrienko, J.-E. Ekberg, A.-R. Sadeghi, and N. Asokan. Key attestation from trusted execution environments. In A. Acquisti, S. Smith, and A.-R. Sadeghi, editors, *3rd International Conference on Trust and Trustworthy Computing*, volume 6101 of *LNCS*, pages 30–46. Springer, Berlin Germany, 2010.
19. C. Marforio, N. Karapanos, C. Soriente, K. Kostiainen, and S. Capkun. Secure enrollment and practical migration for mobile trusted execution environments. In *Third ACM workshop on Security and privacy in smartphones & mobile devices*, pages 93–98, Berlin, Germany, november 2013. ACM Press.

20. A.-R. Sadeghi, C. Stüble, and M. Winandy. Property-based TPM virtualization. In T.-C. Wu, C.-L. Lei, V. Rijmen, and D.-T. Lee, editors, *11th Information Security Conference*, volume 5222 of *LNCS*, pages 1–16, Taipei, Taiwan, september 2008. Springer.
21. Trusted Computing Group. TPM main specification. http://www.trustedcomputinggroup.org/resources/tpm_main_specification, 2015.

A Our transfer protocol in HLPSL

```
role sdSrc (SrcTEE, SDsrc, SDtgt, SP: agent,
            PKSrcTEE, PKSDsrc, PKSDtgt, PKSP: public_key,
            SDCred: text,
            SND, RCV: channel (dy))
    played_by SDsrc def=
    local
        State: nat,
        EncSD: text,
        KProxy: public_key
    init State:=0
    transition
    0.State=0 /\ RCV(start) =|> EncSD':={SDCred}_PKSDsrc /\ State':=1
    1.State=1 =|> SND(EncSD)
    /\ secret(SDCred,transfer,{SDsrc, SP, SDtgt})
    /\ equal({EncSD}_KProxy, {SDCred}_PKSDtgt)
end role

role src (SrcTEE, SDsrc, TgtTEE, TEEAdmin, SP: agent,
          PKSrcTEE, PKSDsrc, PKTgtTEE, PKTEEAdmin, PKSP : public_key,
          SK : symmetric_key,
          SND, RCV: channel (dy))
    played_by SrcTEE def=
    local
        State : nat,
        TimeStamp,EncSD: text,
        Ack: message,
        KProxy: public_key
    init State := 0
    transition
    0.State = 0 /\ RCV(EncSD) =|> State':= 1 /\ TimeStamp':= new()/\SND(SrcTEE.
          TEEAdmin.PKSP.PKSDsrc. PKSrcTEE. PKTgtTEE.{PKSP. PKSDsrc. PKSrcTEE.
          PKTgtTEE.TimeStamp'}_inv(PKSrcTEE))
    1.State= 1 /\ RCV(SrcTEE.TEEAdmin.KProxy.{PKSP.PKSrcTEE. PKTgtTEE}_inv(PKSP)
          .{KProxy.{ PKSP.PKSrcTEE. PKTgtTEE. TimeStamp}_inv(PKSP)}_inv(
          PKTEEAdmin)) =|> State':= 2
    2.State=2/\SND(SrcTEE.TgtTEE.{{EncSD}_KProxy.PKSP}_SK) =|> RCV({SrcTEE.
          TgtTEE.Ack.TimeStamp}_SK)
end role

role teeAdmin (SrcTEE, TgtTEE, TEEAdmin, SP: agent,
               PKSrcTEE, PKTgtTEE, PKTEEAdmin, PKSP: public_key,
               SND, RCV: channel (dy))
    played_by TEEAdmin def=
    local
        State : nat,
        SDCreate, Param, Ack: message,
        TimeStamp: text,
        PKSDsrc, KProxy: public_key
    init State := 0
    transition
    0.State=0 /\ RCV(TEEAdmin.SrcTEE.PKSP.PKSDsrc.PKSrcTEE.PKTgtTEE.{PKSP.
          PKSDsrc.PKSrcTEE. PKTgtTEE.TimeStamp}_inv(PKSrcTEE))=|> State':= 1 /\ SND
          (TEEAdmin.SP.PKSDsrc.PKSrcTEE. PKTgtTEE)
    1.State=1 /\ RCV(TEEAdmin.SP .SDCreate .PKTgtTEE) =|> State':=2 /\ TimeStamp':=
          new()/\SND(TEEAdmin. TgtTEE. SDCreate. {SDCreate.TimeStamp'}_inv(
          PKTEEAdmin))
    2.State=2 /\RCV(TEEAdmin. TgtTEE.Ack.Param.{ Ack.Param.TimeStamp}_inv(PKTgtTEE
          )) =|> State':=3 /\ SND(TEEAdmin.SP.Param)
    3.State=3 /\RCV(TEEAdmin.SP.KProxy.{PKSP.PKSrcTEE. PKTgtTEE. TimeStamp}_inv(
          PKSP))=|> SND(TEEAdmin. SrcTEE.KProxy.{ PKSP.PKSrcTEE. PKTgtTEE.
          TimeStamp}_inv(PKSP).{ KProxy.{PKSP.PKSrcTEE. PKTgtTEE.TimeStamp}_inv(
          PKSP)}_inv(PKTEEAdmin))
end role

role sp(TEEAdmin, SP: agent,
        PKTEEAdmin, PKSP: public_key,
        SND, RCV: channel (dy))
    played_by SP def=
    local
        State : nat,
        SDCreate, Param: message,
        TimeStamp: text,
```

```
60          PKSrcTEE, PKTgtTEE, PKSDsrc, PKSDtgt, KProxy: public_key
61      init State := 0
62      transition
63      0.State=0 /\ RCV(SP.TEEAdmin.PKSDsrc.PKSrcTEE.PKTgtTEE) =|> State':=1 /\ SND(
            SP.TEEAdmin.SDCreate.PKTgtTEE)
64      1.State=1/\RCV(Param) =|> TimeStamp':=new()/\PKSDtgt':=new()/\ KProxy':=new()/\
            SND(KProxy'.{PKSP.PKSrcTEE.PKTgtTEE.TimeStamp'}_inv(PKSP))
65  end role
66
67  role tgt(SrcTEE, TgtTEE, TEEAdmin, SDTgt: agent,
68           PKTgtTEE, PKTEEAdmin, PKSP, PKSDtgt: public_key,
69           SK : symmetric_key,
70           SND, RCV: channel (dy))
71      played_by TgtTEE def=
72      local
73          State : nat,
74          TimeStamp, EncSD: text,
75          SDCreate, Ack, Param: message,
76          KProxy: public_key
77      init State := 0
78      transition
79      0.State=0/\RCV(TgtTEE.TEEAdmin.SDCreate.{SDCreate.TimeStamp}_inv(PKTEEAdmin))
            =|> State':=1/\TimeStamp':=new()/\SND(TgtTEE.TEEAdmin.Ack.Param.{Ack.
            Param.TimeStamp'}_inv(PKTgtTEE))
80      1.State=1/\RCV(TgtTEE.SrcTEE.{{EncSD}_KProxy.PKSP}_SK)=|>State':=2/\TimeStamp
            ':=new()/\SND(TgtTEE.SrcTEE.{Ack.TimeStamp'}_SK)
81      2.State=2 =|>SND(TgtTEE.SDTgt.{EncSD}_KProxy)
82  end role
83
84  role sdTgt(TgtTEE, SDTgt: agent,
85             PKTgtTEE, PKSDsrc, PKSDtgt: public_key,
86             SND, RCV: channel (dy))
87      played_by SDTgt def=
88      local
89          State: nat,
90          EncSD: text,
91          KProxy: public_key
92      init State:=0
93          transition
94      0.State= 0 =|> RCV({EncSD}_KProxy)
95  end role
96
97  role session(SDsrc, SDtgt, SrcTEE, TgtTEE, TEEAdmin, SP: agent,
98               PKSDsrc,PKSDtgt,PKSrcTEE,PKTgtTEE,PKTEEAdmin,PKSP: public_key,
99               SK : symmetric_key,
100              SDCred: text)
101     def=
102     local S0, R0, S1, R1, S2, R2, S3, R3, S4, R4, S5, R5 : channel (dy)
103     composition
104     sdSrc (SrcTEE, SDsrc, SDtgt, SP, PKSrcTEE, PKSDsrc, PKSDtgt, PKSP, SDCred, S0, R0)
105     /\ src (SrcTEE, SDsrc, TgtTEE, TEEAdmin, SP, PKSrcTEE, PKSDsrc, PKTgtTEE,
                PKTEEAdmin, PKSP, SK, S1, R1)
106     /\ teeAdmin (SrcTEE, TgtTEE, TEEAdmin, SP, PKSrcTEE, PKTgtTEE, PKTEEAdmin,
                PKSP,S2, R2)
107     /\ sp (TEEAdmin, SP, PKTEEAdmin, PKSP, S3, R3)
108     /\ tgt (SrcTEE, TgtTEE, TEEAdmin, SDtgt, PKTgtTEE, PKTEEAdmin, PKSP, PKSDtgt,
                SK, S4, R4)
109     /\ sdTgt(TgtTEE, SDtgt, PKTgtTEE, PKSDsrc, PKSDtgt, S5, R5)
110 end role
111
112 role environment()
113   def=
114   const
115       sdsrc, sdtgt, srctee, tgttee, teeadmin, spagent, i: agent,
116       pksdsrc,pksdtgt,pksrctee,pktgttee,pkteeadmin,pksp,ki: public_key,
117       sk: symmetric_key,
118       transfer : protocol_id,
119       sdcred: text
120   intruder_knowledge={pksrctee, pktgttee, pkteeadmin, pksp, ki, inv(ki)}
121   composition
122   session(sdsrc, sdtgt, srctee, tgttee, teeadmin, spagent, pksdsrc, pksdtgt, pksrctee, pktgttee,
            pkteeadmin, pksp, sk, sdcred)
123   /\ session(sdsrc, sdtgt, i, tgttee, teeadmin, spagent, pksdsrc, pksdtgt, pksrctee, pktgttee,
            pkteeadmin, pksp, sk, sdcred)
124   /\ session(sdsrc, sdtgt, srctee, i, teeadmin, spagent, pksdsrc, pksdtgt, pksrctee, pktgttee,
            pkteeadmin, pksp, sk, sdcred)
125   /\ session(sdsrc, sdtgt, srctee, tgttee, i, spagent, pksdsrc, pksdtgt, pksrctee, pktgttee,
            pkteeadmin, pksp, sk, sdcred)
126 end role
127
128 goal
129     secrecy_of transfer
130 end goal
131
132 environment()
```