Towards User-Friendly Credential Transfer on Open Credential Platforms

Kari Kostiainen[1], N. Asokan[1], and Alexandra Afanasyeva[2]

[1] Nokia Research Center, Helsinki
kari.ti.kostiainen@nokia.com, n.asokan@nokia.com
[2] Saint-Petersburg State University of Aerospace Instrumentation
alra@vu.spb.ru

Abstract. Hardware-based "trusted execution environments" (TrEEs) are becoming widely available and open credentials platforms allow any service provider to issue credentials to such TrEEs. Credential transfer in an open system poses usability challenges: Certain credential issuers may disallow direct credential migration and require explicit credential re-provisioning, and each credential provisioning typically requires separate user authentication. Additionally, the lack of secure user input mechanisms on existing TrEEs makes the required user identity binding to TrEEs challenging. In this paper we present a practical credential transfer protocol that can be implemented using devices available today. Our protocol makes credential transfer user-friendly with delegated, automatic re-provisioning, and can be integrated to a typical device initialization process.

Keywords: security, credential transfer, trusted computing.

1 Introduction

Traditional credentials have well-known drawbacks. User memorizable passwords suffer from bad usability and are vulnerable to phishing and dictionary attacks. Software-based credentials, such as secret keys stored on the device, may provide better usability and security, but can be compromised by exploiting security vulnerabilities inherently present in most modern operating systems. Hardware-based credentials, like dedicated security tokens, provide higher level of security, but are too expensive for most service providers, and due to their typical single-purpose nature force users to carry multiple tokens with them.

During the past decade hardware-based "trusted execution environments" (TrEEs) have started to become widely available in various commodity devices. Many current mobile phones support integrated security architectures like M-Shield [24] and TrustZone [1] that augment the mobile device central processing unit with secure storage and isolated execution. Additionally, many mobile devices are equipped with fixed or removable security elements, such as security enhanced memory cards (see e.g. [17]) and UICCs (Universal Integrated Circuit Card, used for cellular authentication), that provide similar hardware-based isolation. Many personal computers are equipped with a Trusted Platform Module

(TPM) [25] which is a standardized security element that provides secure storage and predefined cryptographic operations. When used with suitable central processing unit TPMs can also provide isolated execution for arbitrary security-sensitive code [16]. Using such TrEEs it is possible to implement credentials that combine good usability of software-based credentials with the higher level of security traditionally only provided by dedicated security tokens.

Some TrEE-based "credential platforms" are closed systems. For example installing new credentials to an UICC typically requires approval from the TrEE owner, i.e. the mobile network operator. On-board Credentials [14] and Trusted Execution Module [6] are examples of *open* credential platforms in which *any* service provider can develop new credential types and install them to devices without prior agreement with the TrEE issuer or manufacturer.

Users should be able to transfer credentials from one compliant TrEE to another. The common use case is device replacement. Assume that Alice has several credentials provisioned to her mobile phone. When Alice buys a new phone, the already provisioned credentials should be made available to the new device as easily as possible. When TrEEs are equipped with a certified key pair, a straightforward credential transfer approach would be to first validate the target device certificate within the source TrEE and then encrypt all the TrEE-resident credentials using the target device public key. In practice, credential transfer is more challenging due to following reasons.

First, while some issuers may allow their credentials to be copied from one compliant TrEE to another, we take a practical stand and assume that others may require explicit credential re-provisioning to the target device from the original credential issuer. In a closed credential system such re-provisioning is easy. Typically, the user has to authenticate himself only towards the TrEE owner which controls all credential provisioning. In an open credential platform, such credential re-provisioning poses usability challenges, since each credential provisioning operation typically requires user authentication with respect to that issuers's domain, and thus having to re-provision credentials from multiple different issuers forces user to perform multiple user authentication operations, e.g. when a new device is taken into use. Such an user experience is clearly not optimal.

Second, validating the target device TrEE certificate only guarantees that copyable credentials are transfered from one compliant TrEE to another. To ensure that the credentials are transfered to a TrEE belonging to the *correct user*, the device certificate, or the credentials themselves, must be bound to the user identity. In case of commodity devices such user identity binding must be done by the user himself, since user identities are not known at the time of device manufacturing and can change during device lifetime. TrEEs on existing commodity devices typically lack secure (user) input mechanisms, i.e. data input to the TrEE is typically possible only via the possibly compromised device operating system, which makes secure user binding challenging.

Third, the user may not have both the source and the target device in his possession at the same time. The user may, e.g. during device replacement, have

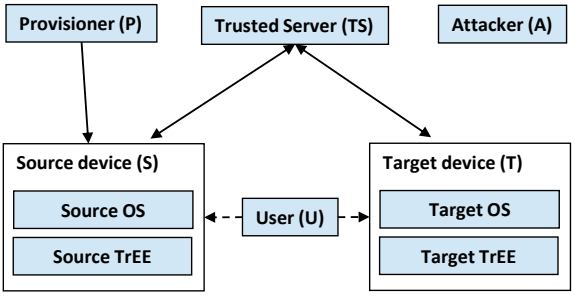

Fig. 1. Credential transfer system model

to give away his old device before he gets the new device, and thus the target device public key cannot be imported to the source device easily.

In this paper we address the problem of user-friendly credential transfer in open credential platforms. We focus on embedded TrEEs on mobile devices, although most of the discussion applies to other embedded TrEE types as well. We present a credential transfer protocol that (1) utilizes delegated, automatic re-provisioning for non-transferable credentials, and (2) handles user identity binding by relying on the "trust on first use" principle to mitigate the lack of secure user input mechanisms. The main contribution of this paper is not so much in the technical novelty of the proposed protocol, but rather in identifying the problem and presenting a practical solution that can be (a) implemented using commodity devices available today and (b) integrated into a typical mobile device initialization process for optimal user experience. We have formally validated a relevant subset of our protocol using AVISPA [26] model-checking security protocol validation tool.

2 Assumptions and Requirements

Assumptions. The entities involved in credential transfer are shown in Fig. 1. Provisioner P issues credentials to a source device S from which the credentials can be transfered to a target device T with the help of a trusted server TS.

We assume that devices S and T have an embedded TrEE with a (statistically) unique asymmetric key pair PK/SK. The key pair has been issued a certificate Cert by a trusted authority (e.g., the device manufacturer). Additionally, the TrEEs are equipped with a symmetric key K that can be used for local encryption, or sealing. TrEEs do not have persistent secure storage in addition to these fixed keys. We assume that only trusted code (e.g., code signed by the device manufacturer) can be executed within the TrEE and can access TrEE-resident keys. A trust root, such as a hash of the trusted authority public key, has been installed within the TrEEs of S and T during manufacturing.

We assume that devices S and T have an operating system level security framework using which access to the TrEE on the devices can be limited to trusted operating system level software components only (e.g., processes with a

required permission). Integrity of the operating system security framework itself may be enforced with TrEE-based secure boot.

A direct and secure communication path between the user and the TrEE is often denoted "trusted user interface" (see e.g. [9] for more information). Despite of promising research prototypes [23] TrEEs on existing commodity mobile devices typically do not support trusted user interfaces or any other data input mechanisms that are not controlled by the possible compromised OS. Thus we assume that the communication between the user and the TrEE is always mediated by the device operating system.

We assume that the mobile devices S and T have a device initialization process that is automatically triggered when the device is booted for the first time (or after device reset operation). As a part of this device initialization the user is asked to log in (e.g., with username-password) to services provided by the mobile platform provider (e.g., Google services for Android devices or Ovi services for Nokia devices).

We assume that the trusted server TS also has an asymmetric key pair $\mathsf{PK}_{\mathsf{TS}}/\mathsf{SK}_{\mathsf{TS}}$ and a certificate $\mathsf{Cert}_{\mathsf{TS}}$ issued by the same trusted authority. A similar trust root (e.g., hash of trusted authority public key) is fixed in TS as well.

We assume an open credential provisioning model in which credentials can be provisioned from any service provider to the TrEEs. The actual provisioning protocol can be provisioner specific, but we assume that each credential provisioning operation always requires both *device* and *user* authentication. We assume that the device authentication is based on the above mentioned TrEE certificates. Each credential issuer may have it's own user authentication mechanism, but we assume that user authentication always requires some user input, such as entering a "provisioning password" to the device. Due to previously mentioned lack of trusted user interface, the provisioning user authentication is always handled by operating system level software component.

We address two possible credential migration policies: (1) *copyable* credentials can be transfered directly from one compliant TrEE to another and (2) *non-transferable* credentials must be re-provisioning from the original issuer with user authentication.

Attacker model. We assume that the attacker A can read and modify any network traffic between P, S, TS and T based on the Dolev-Yao model [8]. The attacker cannot read or modify keys stored on, or read or modify any program execution that takes place within the TrEEs of S and T or on TS. We assume that the attacker may be able to compromise the operating system of S or T at runtime, and thus read and modify any program execution that takes place on the operating system of S or T. The attacker may have one or more devices with a compliant TrEE.

Requirements. We define single functional requirement for credential transfer:

– *No additional user interaction.* Transferring all credentials from S to T should require no additional user interaction besides the normal mobile device initialization process.

We define two security requirements:

– *Credential secrecy.* The attacker must not be able to read or modify any credentials during transfer.
– *Credential fidelity.* Credentials should be transfered only to a device belonging to the same user that the credentials were originally provisioned to.

The credential transfer protocol should guarantee "forward fidelity", i.e. the attacker should not be able to transfer securely provisioned credentials to a device that belongs to a different user if the OS gets compromised after the provisioning. (This requirement is similar to "forward secrecy" in key agreement protocols [7], i.e. the attacker should not be able to determine previously established session keys if the long-term keys used in the key agreement get compromised later.)

Since we assume that the provisioning user authentication is always handled by operating system level component, due to lack of trusted user interface on existing commodity devices, and that the attacker may be able to compromise the device OS, the credential provisioning phase is vulnerable to a man-in-the-middle attacker that can cause a credential to be provisioned to a wrong device, and thus the credential transfer solution cannot credential fidelity for credentials that are provisioned after the OS compromise.

3 Credential Transfer Protocol

The rationale behind the credential transfer protocol is as follows. User identity installation is done by using the "trust on first use" principle. During device first boot, or after device reset, when the device operating system is in an uncompromised state, the password from the normal mobile device initialization process is installed to the TrEE of the device. The installed password is bound to the credentials inside the device TrEE during each credential provisioning. The actual credential transfer happens in three phases. During credential backup, all copyable credentials are copied securely from S to TS. For each non-transferable credential, S creates a *delegation token*. During credential recovery, TS verifies that the same user identity has been installed to T, and if this is the case, all

Table 1. Notations used in credential transfer protocol

c ← Enc(PK, d)	Ciphertext c from encryption on data d using public key PK.
d ← Dec(SK, c)	Plaintext d from decryption on ciphertext c using private key SK.
s ← Sign(SK, d)	Signature s on data d using private key SK.
c ← Seal(d)	Ciphertext c from authenticated encryption on data d with key K.
d ← Unseal(c)	Plaintext d from authenticated decryption on ciphertext c with key K.
m ← MAC(k, d)	Keyed message authentication code m over data d using key k.

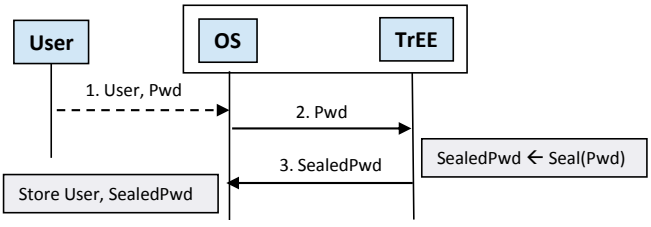

Fig. 2. User identity installation

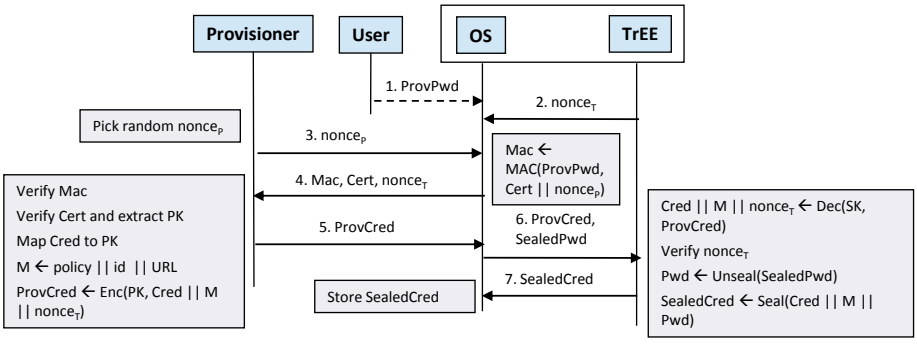

Fig. 3. Credential provisioning

copyable credentials and delegation tokens are transfered from TS to T. In re-provisioning phase, using the delegation tokens, T can automatically fetch, or re-provision, all the non-transferable credentials from the original provisioners without explicit user authentication towards each provisioner.

Table 1 lists the algorithmic notations used in the protocol description. The rest of the notations used are explained in the following text.

Identity installation. When the user starts his device for the first time, user identity should be installed to the TrEE of the device (see Fig. 2).

1. The user is asked to enter a username User and a password Pwd.
2. An operating system level software component forwards Pwd to the TrEE. Within the TrEE the password is locally sealed.
3. The resulting encryption SealedPwd is stored on the operating system side together with User.

Operating system level security mechanism, e.g. permission based access control, is used to enforce that only a trusted software component can perform this initialization only when the device is taken into use for the first time.

Provisioning. The credential provisioning protocol details may vary between different credential issuers, but we assume that the basic principles are always

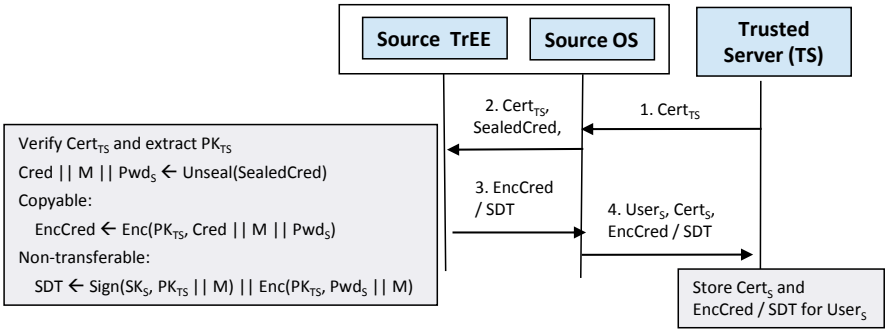

Fig. 4. Credential backup (transfer from the source device S to the trusted server TS)

the same: each credential provisioning requires both device and user authentication. Example provisioning is shown in Fig. 3. We assume that the user and the provisioner share provisioning password ProvPwd from prior out-of-band communication, such as service registration email.

1. The mobile device requests ProvPwd from the user.
2. The TrEE picks a random $\mathsf{nonce_T}$ and returns that to the operating system component on the mobile device.
3. Provisioner picks a random $\mathsf{nonce_P}$ and sends this to the device. An operating system level software component on the device calculates keyed message authentication code Mac using ProvPwd. $\mathsf{nonce_P}$ and the device certificate Cert is included to the message authentication code to bind the user authentication to the device authentication.
4. The device sends Mac, Cert and $\mathsf{nonce_T}$ to the provisioner. The provisioner verifies Mac using previously picked $\mathsf{nonce_P}$ and ProvPwd. The provisioner also verifies Cert to make sure that the credential is provisioned to a compliant TrEE. The provisioner stores a mapping between the provisioned credential Cred and PK. This mapping is later needed for automated re-provisioning. The provisioner encrypts the provisioned credential Cred, credential metadata M and $\mathsf{nonce_T}$ using PK. The credential metadata M includes migration policy, credential identifier and re-provisioning URL for non-transferable credentials.
5. The resulting encryption ProvCred is sent to the device.
6. ProvCred is loaded to the TrEE together with SealedPwd. Inside the TrEE, both of these encryptions can be recovered and Cred is sealed for local storage. The user identity password Pwd is included to the seal to bind the installed credential to current user identity. The freshness of $\mathsf{nonce_T}$ is also verified.
7. The resulting encryption SealedCred is stored on the operating system side.

Credential backup. Credential transfer from the source device S to the trusted server TS is described in Fig. 4.

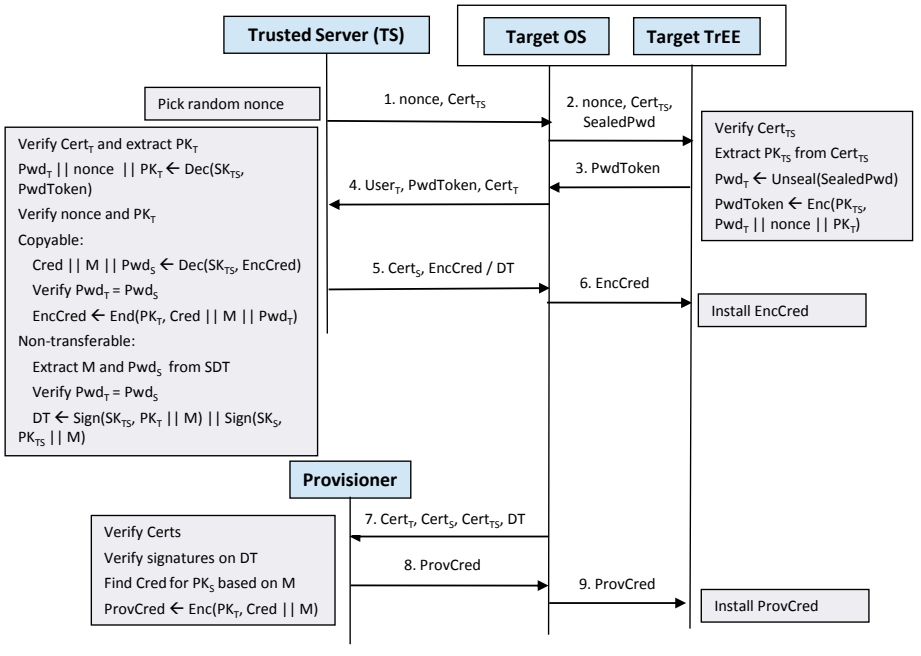

Fig. 5. Credential recovery (transfer from trusted server TS to target device T)

1. The trusted server sends its certificate $\mathsf{Cert_{TS}}$ to the source device.
2. The source device loads $\mathsf{Cert_{TS}}$ and each SealedCred to the TrEE. The source device verifies $\mathsf{Cert_{TS}}$ inside the TrEE with respect to the pre-installed trust root to make sure that credentials are transfered only via the trusted server. The source device unseals each SealedCred and in case of copyable credential the source device encrypts it using the public key of the trusted server $\mathsf{PK_{TS}}$ which can be extracted from $\mathsf{Cert_{TS}}$. The resulting encrypted credential EncCred includes the user identity password of the source device $\mathsf{Pwd_S}$. If the credential is non-transferable a server delegation token SDT is created. This token contains $\mathsf{PK_{TS}}$ and M signed with the private key of the source device $\mathsf{SK_S}$ concatenated with source device user identity password $\mathsf{Pwd_S}$ and M encrypted for the trusted server.
3. Each encrypted credential EncCred and SDT for each non-transferable credential are returned to the operating system of the source device.
4. The operating system sends these to the trusted server together with username $\mathsf{User_S}$ and $\mathsf{Cert_S}$ of the source device. The trusted server stores all received data items.

Credential recovery. Credential transfer from the trusted server TS to the target device T is described in Fig. 5. Prior to this, the user identity installation must be done to T as described before.

1. The trusted server picks a random nonce and sends this together with the server certificate $\mathsf{Cert_{TS}}$ to the target device.
2. The target device loads these to the TrEE with the local SealedPwd. Inside the target device TrEE $\mathsf{Cert_{TS}}$ is verified using the pre-installed trust root and $\mathsf{PK_{TS}}$ is extracted from the certificate. The target device creates a password token PwdToken by encrypting the user identity password of the target device $\mathsf{Pwd_T}$, nonce and $\mathsf{PK_T}$ using the public key of the server $\mathsf{PK_{TS}}$.
3. PwdToken is returned to the target device operating system.
4. The target device sends PwdToken, $\mathsf{User_T}$ and $\mathsf{Cert_T}$ to the trusted server. The trusted server verifies $\mathsf{Cert_T}$ to make sure that the credentials are transfered only to a compliant TrEE. The server decrypts PwdToken and verifies that the received nonce matches the one picked earlier and that the public key $\mathsf{PK_T}$ inside the password token matches the one in $\mathsf{Cert_T}$. For each copyable credential the trusted server does as follows: EncCred received from source device is decrypted using the private key of the server. The server verifies that the user identity password $\mathsf{Pwd_T}$ recovered from PwdToken matches $\mathsf{Pwd_S}$ recovered from EncCred. If this is the case, the target device belongs to the same user as the source device and the server encrypts the credential Cred using the public key of the target device $\mathsf{PK_T}$. The resulting encryption is denoted EncCred. For each non-transferable credential the server decrypts $\mathsf{Pwd_S}$ and M from SDT. The server verifies that $\mathsf{Pwd_T}$ received from the target device in PwdToken matches $\mathsf{Pwd_S}$ and if this is the case the server creates a new delegation token DT for the target device. This token includes a signature over the target device public key $\mathsf{PK_T}$ and M using the private key of the server $\mathsf{SK_{TS}}$ and the original signature made by the source device from SDT.
5. The server sends EncCred for each copyable credential, DT for each non-transferable credential and Certs to the target device.
6. The target device can install (decrypt and locally seal) each EncCred inside it's TrEE.
7. Using each received delegation token DT the target device can fetch non-transferable credentials from the original provisioners. The target device sends the certificates of all three devices (Certs, $\mathsf{Cert_{TS}}$ and $\mathsf{Cert_T}$) and a delegation token DT to a provisioner of a non-transferable credential. The URL of the provisioning server can be extracted from the credential metadata M. The provisioner verifies these three certificates and checks that DT has correct signature chain, i.e. the source device has delegated the trusted server and that the trusted server in turn has delegated the target device. The delegation token signatures include credential metadata M that includes credential identifiers. The provisioner checks that these credential identifiers match $\mathsf{PK_S}$ (can be extracted from Certs). If all the above mentioned conditions hold true, then the original credential Cred can be provisioned to the target device without explicit provisioning user authentication and the provisioner constructs ProvCred packages for the target device.
8. ProvCred is sent to the target device.
9. The target device can install ProvCred as described before.

4 Requirement Analysis

Based on our requirements the credential transfer should require no additional user interaction besides the normal mobile device initialization process. Credential backup can be automated so that credentials are copied from the source device to the trusted server automatically whenever there is a change in one of the credentials. Credential recovery can be run automatically after normal device login operation. Thus, no further user interaction besides the normal login operation is needed.

The attacker must not be able to read or modify credentials during transfer (credential secrecy). This is enforced using common public key infrastructure mechanisms. The credentials are only encrypted to a trusted server or to a compliant TrEE, and the encryption keys are validated with device certificates and using trust roots fixed to device TrEEs during manufacturing. Only trusted code is executed on the trusted server and compliant TrEEs, and trusted code will not leak credentials outside these trusted environments.

The credentials should be transfered only to a device belonging to the same user that the credentials were originally provisioned to (credential fidelity). This is enforced with a password, and thus meeting this requirement relies on the assumption that the attacker will not learn (or be able to guess) the correct password, even if he manages to compromise the operating system on the source or target device after device initialization process. User chosen and memorizable password have well known security issues, most notably vulnerability to offline dictionary attacks and phishing. In our protocol the password enforcement is done on the trusted on-line server, and thus common dictionary attack prevention techniques, such as throttling, can be applied. Our approach relies on the assumption that the user enters the password only to a trustworthy device initialization software on the device when the device is booted for the first time. If the user enters the password to any other (possible malicious) software on the device the attacker can learn the password.

Operating system level enforcement mechanisms is used to ensure that a malicious application will not replace the already installed user identity. If the attacker manages to compromise the source device operating system, he may replace currently installed user identity password with a freely chosen one and all the credentials provisioned *after* the operating system compromise will be bound to the attacker chosen password. As a result the attacker may transfer such credentials to his device. This limitation of the solution is acceptable, since an attacker that is able compromise the operating system can act as a man-in-the-middle in original provisioning protocol and direct credential provisioning to a compliant attacker device anyway.

5 Protocol Validation

We have validated (the core subset of) our credential transfer protocol using AVISPA [26]. AVISPA is an automated security protocol validation tool. Security protocols are modeled using High-Level Protocol Specification Language

(HLPSL) and the tool automatically translates such protocol models into an equivalent infinite state transition system that is then input to validation back-ends. The validation back-ends search the transition system for states that represent attacks, i.e. states and properties defined in terms of HLPSL. All the back-ends analyze protocols by considering the Dolev-Yao model [8] of an active adversary that controls the network but cannot break cryptography.

We have modeled the credential transfer protocol described in the this paper with the following two restrictions. First, our current model only addresses copyable credentials. Second, our current model does not cover operations that involve user interaction, i.e. user identity installation and provisioning. Instead we make the simplifying assumptions that (1) the user identity password has already been securely installed to the TrEE of the source device (assuming trust on first use), (2) the credential has already been provisioned to the TrEE of the source device, and (3) the same user identity password has securely been installed to the TrEE of the target device (again, trust on first use). Appendix A shows listing of the protocol validation model in HSPSL. We see that extending the model to cover the migration policy of non-transferable credentials is fairly straightforward. Modeling user identity installation and provisioning, i.e. operations that involve user interaction, pose more challenges, especially under our assumptions and threat model.

In our protocol model each entity is described as a separate role. Each role has a set of input parameters (e.g., other roles to communicate with, and keys and other data used within that role). The roles communicate over channels that the adversary can fully control. The roles are combined into a session construct, and sessions can be instantiated with constant parameters that e.g. define the keys that each role will use in that session.

HLPSL offers two mechanisms for modeling security requirements. Declaration `secret(d, id, r)` defines a security goal with identifier `id` that defines that data `d` should remain secret between set of roles `r`. Typically, authentication security requirements are modeled with `witness` and `request` declarations. For our simplified credential transfer model, both security requirements (SR1 and SR2) can be defined in one security goal `sr`. Declaration `secret(Cred, sr, {SourceTree, Server, TargetTree})` at the end of `role_SourceTree` defines that the transfered credential `Cred` should remain secret between the TrEE of the source device, the trusted server and the TrEE of the target device. Since the credential is not revealed to other parties, SR1 is met. Since the credential is not transferred to TrEE of any other compliant device, SR2 is met. We model our attacker model so that the adversary plays role of `role_TargetTree` with the knowledge of matching certified key pair. The correct user identity password `pwd` is not given to target device TrEE played by the adversary.

We have successfully validated our protocol model with three ASVISPA back-ends (OFMC, CL-AtSe and SATMC). The validation of the model can be easily re-produced from the AVISPA tool web interface.[1]

[1] `http://www.avispa-project.org/web-interface/`

6 Discussion

In our credential transfer protocol the user authentication is based on a password that is installed after the first boot ("trust on first use" approach). This solution meets our security requirements, provides nice user experience (assuming integration to normal device initialization process), and can be implemented with commodity devices available today. The drawbacks of this approach are vulnerability to phishing and the fact that the server has to be fully trusted.

Alternatively, the user authentication could be based on a full-length key that would is installed to the TrEE after first boot and copied e.g. to a removable memory card. First part of the key would be used for encrypting the credentials for the server and second part would be user for user authentication. Such an approach would not have the problem of phishing and would not require fully trusted server. The drawbacks of this approach would be inconvenient user interaction model, since the memory card would have to be removed from the device after first boot to prevent the secret leaking to the OS that may get compromised later. In practice the user would need a dedicated memory card for credential transfer.

Another approach would be to rely on availability of "trusted user interface" [9]. Such an approach would maintain the user interaction model of our solution. Additionally provisioning user authentication could be TrEE-based, and since the TrEE could reliably distinguish between user and application provided input, a malicious application could not replace the already installed password, and the attacker could not fool the system into transferring credentials provisioned after the device OS compromise into an unauthorized device. Trusted user interface would make phishing more difficult as well, although studies have shown that users tend to ignore security most indicators [20]. In practice, despite of promising research prototypes [23], TrEEs on existing mobiles do not support such trusted user interfaces.

Another alternative would be to use a "trusted external interface". Near Field Communication (NFC) enabled mobile devices are currently becoming increasingly popular [11] and due to many security-related NFC use cases device manufacturers are integrating NFC chips to device TrEEs. If such an interface would be available, credential transfer could be based on demonstrative identification [2], i.e. the user would touch and the target device with the source device, and the source device could transfer (and delegate) credentials directly to the target device. In practice, such devices are not yet widely available.

Finally, on mobile devices, a natural approach would be to use presence of the correct UICC for credential transfer authentication. During first boot, UICC identifier could be installed to the TrEE and that could be bound to all provisioned credentials. The target device could prove the presence of the same UICC to the trusted server e.g. using Generic Bootstrapping Architecture [12]. Such an approach would require no additional user interaction besides normal device switch operations. The drawback would be that such an approach would no meet our security model. Since UICCs are not securely integrated to TrEEs, a compromised OS could direct credential transfer to an unauthorized device.

Table 2. Comparison of user authentication alternatives for credential transfer

user authentication mechanism	vulnerability to OS compromise	vulnerability to phishing	extra user interaction	requires fully trusted server	device available today
trust on first use	no *	vulnerable	no	yes	yes
trust on first use (memory card)	no *	no	yes	no	yes
trusted user interface	no	less vulnerable	no	yes	no
trusted external interface	no	no	yes (touch)	no	no
UICC presence	yes	no	no	yes	yes

*) Subject to the assumption of OS not being compromised during first use.

We summarize key properties of above discussed user authentication alternatives in Table 2.

In this paper we have addressed two credential migration policies: copyable and non-transferable credentials. A third natural migration policy would be "movable credentials", i.e. credentials that are allowed to exist in one (or predefined number of) compliant TrEEs at the same time. The notable difference to copyable credentials is that credential deletion or disabling from the source device should be possible. Deleting or disabling credentials from a TrEE requires replay-protection support from the TrEE hardware, e.g. in the form of monotonic secure counter or non-volatile secure memory. Many existing TrEEs on mobile device do not provide such support (see e.g. [21] for rationale), and thus enforcement of movable credential migration policies must be done on the infrastructure back-end. Additionally, in many cases the credential issuers want to control credential migration even tough the end user devices would support replay protection.

Defining movable credentials as non-transferable, and thus mandating explicit re-provisioning, is one way to allow the credential provisioner to control the number of the devices to which the credential can be moved. Alternatively, such migration control could be implemented on the infrastructure back-end by verifying device identity during on-line credential usage. The drawback of such approach is that it requires changes to the actual credential usage protocols. Our approach allows the credential issuers to control credential replication by the means of explicit re-provisioning, which requires changes only to credential provisioning but not the credential usage protocols.

7 Related Work

Credential transfer between TrEEs has been studied primarily in the context of Trusted Platform Modules (TPMs). The TPM specifications define key migration commands [25]. When a new TPM based key is created, the calling application may define a migration password. The key may be migrated only by applications that know this password. The TPM specifications also define concepts of Migration Authority (MA) and Migration Selection Authority (MSA)

[25] that in principle can control to which devices TPM credentials are migrated to. The specification do not, however, define how user binding is done for such credential migration.

Several extensions and improvements have been proposed to TPM migration commands. [10] presents a protocol for migrating of TPM-based web authentication credentials using existing TPM commands. [15] proposes extensions to TPM commands. In [5] and [19] credential migration is studied in the context of digital rights management. [3] describes a mechanism for TPM virtualization and how such virtual TPMs could be migrated from one device to another. Property-based TPM virtualization and migration is described in [18]. Migration of Mobile Trusted Modules (MTM) is described in [22]. Temporary disabling of TrEE-based credentials on mobile devices has been studied in [13]. However, none of these works address the problem we are interested in, i.e. user-friendly re-provisioning of non-transferable credentials and secure user binding to prevent credentials from being transfered to a compliant but incorrect TrEE.

Our approach requires a fully trusted server. Boyen has proposed a scheme to protect credentials on a server with a single user memorizable password that is used for both access authentication and encryption in a way that the password is not revealed to the server during access authentication [4]. The untrusted server cannot recover password-encrypted credentials assuming the the credentials do not have recognizable structure, and thus brute force attacks against password-encrypted credentials are not possible. This assumption would not, however, hold true for the credential platforms we are addressing [14,6], since the credentials do have recognizable structure needed e.g. for TrEE-internal access control enforcement (and TrEE-external credential metadata).

8 Summary

In this paper we have addressed the problem of credential transfer between TrEEs on mobile devices. Credential transfer is challenging assuming an open credential provisioning model in which each credential issuer has its own user authentication domain and migration policies. Additionally, the lack of secure user input mechanisms in existing TrEEs make credential transfer challenging. The contribution of this paper is a novel and practical credential transfer protocol that requires minimal user interaction and can be implemented using commodity devices available today.

References

1. ARM. Trustzone-enabled processor, `http://www.arm.com/products/processors/technologies/trustzone.php`
2. Balfanz, D., Smetters, D.K., Stewart, P., Wong, H.C.: Talking to strangers: Authentication in ad-hoc wireless networks. In: Proc. Network and Distributed System Security Symposium (NDSS 2002) (2002)

3. Berger, S., Caceres, R., Goldman, K., Perez, R., Sailer, R., van Doorn, L.: vTPM - virtualizing the trusted platform module. In: Proc. 15th Usenix Security Symposium (2006)
4. Boyen, X.: Hidden credential retrieval from a reusable password. In: Proc. 4th International Symposium on Information, Computer, and Communications Security, ASIACCS 2009 (2009)
5. Cooper, A., Martin, A.: Towards an open, trusted digital rights management platform. In: Proc. ACM Workshop on Digital Rights Management, DRM 2006 (2006)
6. Costan, V., Sarmenta, L.F.G., van Dijk, M., Devadas, S.: The trusted execution module: Commodity general-purpose trusted computing. In: Grimaud, G., Standaert, F.-X. (eds.) CARDIS 2008. LNCS, vol. 5189, pp. 133–148. Springer, Heidelberg (2008)
7. Diffie, W., Van Oorschot, P.C., Wiener, M.J.: Authentication and authenticated key exchanges. Des. Codes Cryptography 2 (1992)
8. Dolev, D., Yao, A.C.: On the security of public key protocols. Technical report, Stanford, CA, USA (1981)
9. Fischer, T., Sadeghi, A.-R., Winandy, M.: A pattern for secure graphical user interface systems. In: Bhowmick, S.S., Küng, J., Wagner, R. (eds.) DEXA 2009. LNCS, vol. 5690. Springer, Heidelberg (2009)
10. Gajek, S., Löhr, H., Sadeghi, A.-R., Winandy, M.: Truwallet: trustworthy and migratable wallet-based web authentication. In: Proc. ACM Workshop on Scalable Trusted Computing, STC 2009 (2009)
11. Harrop, P., Das, R.: Nfc-enabled phones and contactless smart cards 2010-2020. Technical report, IDTechEx (2010), `http://www.idtechex.com/research/`
12. Holtmanns, S., Niemi, V., Ginzboorg, P., Laitinen, P., Asokan, N.: Cellular Authentication for Mobile and Internet Services. Wiley, Chichester (2008)
13. Kostiainen, K., Asokan, N., Ekberg, J.-E.: Credential disabling from trusted execution environments. In: Proc. of Nordic Conference in Secure IT Systems, Nordsec 2010 (2010)
14. Kostiainen, K., Ekberg, J.-E., Asokan, N., Rantala, A.: On-board credentials with open provisioning. In: Proc. ACM Symposium on Information, Computer & Communications Security, ASIACCS 2009 (2009)
15. Kühn, U., Kursawe, K., Lucks, S., Sadeghi, A.-R., Stüble, C.: Secure data management in trusted computing. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 324–338. Springer, Heidelberg (2005)
16. McCune, J.M., Parno, B., Perrig, A., Reiter, M.K., Seshadri, A.: Minimal TCB Code Execution (Extended Abstract). In: Proc. IEEE Symposium on Security and Privacy (May 2007)
17. Poitner, M.: Mobile security becomes reality – the mobile security card (2008), `http://www.ctst.com/CTST08/pdf/Poitner.pdf`
18. Sadeghi, A.-R., Stüble, C., Winandy, M.: Property-based TPM virtualization. In: Wu, T.-C., Lei, C.-L., Rijmen, V., Lee, D.-T. (eds.) ISC 2008. LNCS, vol. 5222, pp. 1–16. Springer, Heidelberg (2008)
19. Sadeghi, A.-R., Wolf, M., Stüble, C., Asokan, N., Ekberg, J.-E.: Enabling fairer digital rights management with trusted computing. In: Garay, J.A., Lenstra, A.K., Mambo, M., Peralta, R. (eds.) ISC 2007. LNCS, vol. 4779, pp. 53–70. Springer, Heidelberg (2007)
20. Schechter, S.E., Dhamija, R., Ozment, A., Fischer, I.: The emperor's new security indicators. In: Proc. IEEE Symposium on Security and Privacy, SP 2007 (2007)

21. Schellekens, D., Tuyls, P., Preneel, B.: Embedded trusted computing with authenticated non-volatile memory. In: Lipp, P., Sadeghi, A.-R., Koch, K.-M. (eds.) Trust 2008. LNCS, vol. 4968, pp. 60–74. Springer, Heidelberg (2008)
22. Schmidt, A., Kuntze, N., Kasper, M.: On the deployment of mobile trusted modules. In: Proc. Wireless Communications and Networking Conference, WCNC 2008 (2008)
23. Selhorst, M., Stüble, C., Feldmann, F., Gnaida, U.: Towards a trusted mobile desktop. In: Acquisti, A., Smith, S.W., Sadeghi, A.-R. (eds.) TRUST 2010. LNCS, vol. 6101, pp. 78–94. Springer, Heidelberg (2010)
24. Srage, J., Azema, J.: M-Shield mobile security technology (2005), TI White paper, http://focus.ti.com/pdfs/wtbu/ti_mshield_whitepaper.pdf
25. TCG. TPM Specifications (July 2007), http://www.trustedcomputinggroup.org/resources/tpm_main_specification
26. Viganò, L.: Automated security protocol analysis with the avispa tool. Electronic Notes in Theoretical Computer Science 155, 61–86 (2006)

A HLPSL Protocol Model

```
role role_SourceTree(SourceTree, SourceOs, Server, TargetTree : agent,
                     SourceSealKey, Pwd : symmetric_key,
                     PKSource, PKServerCA : public_key,
                     SND, RCV : channel(dy))
    played_by SourceTree def=
    local
        State : nat,
        SealedPwd, SealedCred : message,
        Cred : text,
        PKServer : public_key
    init
        State := 0
    transition
    1.  State=0 /\ RCV(start) =|>
        State'=1 /\ Cred' := new() /\
        SND(SourceTree.SourceOs.{Cred'.Pwd}_SourceSealKey)
    2.  State=1 /\ RCV(SourceOs.SourceTree.{PKServer'}_inv(PKServerCA).
      {Cred'.Pwd'}_SourceSealKey) =|>
        State'=2 /\ SND(SourceTree.SourceOs.{Cred'.Pwd'}_PKServer') /\
        secret(Cred, sr, {SourceTree, Server, TargetTree})
end role

role role_SourceOs(SourceOs, SourceTree, Server : agent,
                   PKSource, PKDevCA : public_key,
                   SND, RCV : channel(dy))
    played_by SourceOs def=
    local
        State : nat,
        SealedCred, ServerCert, SourceCert, EncCred : message
    init
        State := 0
    transition
    1.  State=0 /\ RCV(SourceTree.SourceOs.SealedCred') =|>
        State'=1
    2.  State=1 /\ RCV(Server.SourceOs.ServerCert') =|>
        State'=2 /\ SND(SourceOs.SourceTree.ServerCert'.SealedCred)
    3.  State=2 /\ RCV(SourceTree.SourceOs.EncCred') =|>
        State'=3 /\ SND(SourceOs.Server.EncCred'.SourceCert)
end role

role role_Server(Server, SourceOs, TargetOs : agent,
                 PKServer, PKDevCA, PKServerCA : public_key,
                 SND, RCV : channel(dy))
```

```
    played_by Server def=
    local
        State : nat,
        Cred, Nonce : text,
        Pwd : symmetric_key,
        PKSource, PKTarget : public_key,
        EncCred : message
    init
        State := 0
    transition
    1.  State=0 /\ RCV(start) =|>
        State':=1 /\ SND(Server.SourceOs.{PKServer}_inv(PKServerCA))
    2.  State=1 /\ RCV(Server.SourceOs.{Cred'.Pwd'}_PKServer.
      {PKSource'}_inv(PKDevCA)) =|>
        State':=2 /\ Nonce':= new() /\
        SND(Server.TargetOs.Nonce'.{PKServer}_inv(PKServerCA))
    3.  State=2 /\ RCV(TargetOs.Server.{Pwd.Nonce.PKTarget'}_PKServer.
      {PKTarget'}_inv(PKDevCA) ) =|>
        State':=3 /\ EncCred' := {Cred.Pwd}_PKTarget' /\
        SND(Server.TargetOs.EncCred')
end role

role role_TargetOs(TargetOs, TargetTree, Server : agent,
                   PKTarget, PKDevCA : public_key,
                   SND, RCV : channel(dy))
    played_by TargetOs def=
    local
        State : nat,
        Nonce : text,
        SealedPwd, ServerCert, PwdToken, EncCred : message
    init
        State := 0
    transition
    1.  State=0 /\ RCV(TargetTree.TargetOs.SealedPwd') =|>
        State':=1
    2.  State=1 /\ RCV(Server.TargetOs.Nonce'.ServerCert') =|>
        State':=2 /\ SND(TargetOs.TargetTree.Nonce'.ServerCert'.SealedPwd)
    3.  State=2 /\ RCV(TargetTree.TargetOs.PwdToken') =|>
        State':=3 /\ SND(TargetOs.Server.PwdToken'.{PKTarget}_inv(PKDevCA))
    4.  State=3 /\ RCV(Server.TargetOs.EncCred') =|>
        State':=4 /\ SND(TargetOs.TargetTree.EncCred')
end role

role role_TargetTree(TargetTree, TargetOs, SourceTree : agent,
                     TargetSealKey, Pwd : symmetric_key,
                     PKTarget, PKServerCA : public_key,
                     SND,RCV : channel(dy))
    played_by TargetTree def=
    local
        State : nat,
        Nonce, Cred : text,
        PKServer : public_key
    init
        State := 0
    transition
    1.  State=0 /\ RCV(start) =|>
        State':=1 /\ SND(TargetTree.TargetOs.{Pwd}_TargetSealKey)
    2.  State=1 /\ RCV(TargetOs.TargetTree.Nonce'.{PKServer'}_inv(PKServerCA).
      {Pwd'}_TargetSealKey) =|>
        State':=2 /\ SND(TargetTree.TargetOs.{Pwd'.Nonce'.PKTarget}_PKServer')
    3.  State=2 /\ RCV(TargetOs.TargetTree.{Cred'.Pwd'}_PKTarget) =|>
        State':=3
end role

role session(Provisioner, SourceOs, SourceTree, Server : agent,
             TargetOs, TargetTree : agent,
             PKSource, PKTarget, PKServer : public_key,
             PKDevCA, PKServerCA : public_key,
```

```
                    SourceSealKey, TargetSealKey : symmetric_key,
                    SourcePwd, TargetPwd : symmetric_key)
    def=
    local
        SND1, RCV1, SND2, RCV2, SND3, RCV3,
        SND4, RCV4, SND5, RCV5 : channel(dy)
    composition
        role_SourceTree(SourceTree, SourceOs, Server,
            TargetTree, SourceSealKey, SourcePwd, PKSource,
            PKServerCA, SND1, RCV1) /\
        role_SourceOs(SourceOs, SourceTree, Server,
            PKSource, PKDevCA, SND2, RCV2) /\
        role_Server(Server, SourceOs, TargetOs, PKServer,
          PKDevCA, PKServerCA, SND3, RCV3) /\
        role_TargetOs(TargetOs, TargetTree, Server,
          PKTarget, PKDevCA, SND4, RCV4) /\
        role_TargetTree(TargetTree, TargetOs, SourceTree,
          TargetSealKey, TargetPwd, PKTarget, PKServerCA, SND5, RCV5)
end role

role environment()
    def=
    const
        provisioner, sourceos, sourcetree, server, targetos, targettree : agent,
        pksource, pktarget, pki, pkserver, pkdevca, pkserverca : public_key,
    sourcesealkey, targetsealkey, isealkey, pwd, pwdi : symmetric_key,
        sr : protocol_id

    intruder_knowledge = {provisioner, sourceos, sourcetree, server,
                          targetos, targettree, pksource, pktarget,
                          pki, inv(pki), pkdevca, pkserverca}
    composition
        session(provisioner, sourceos, sourcetree, server, targetos, i,
                pksource, pki, pkserver, pkdevca, pkserverca,
                sourcesealkey, isealkey, pwd, pwi)
end role

goal
    secrecy_of sr
end goal

environment()
```