Electronic Preprint for Journal of Information Processing Vol.28

Regular Paper

Secure Authentication Key Sharing between Personal Mobile Devices Based on Owner Identity

Hideo Nishimura[1] Yoshihiko Omori[1,a)] Takao Yamashita[1,b)]

Received: June 5, 2019, Accepted: January 16, 2020

Abstract: Public-key-based Web authentication can be securely implemented using modern mobile devices as secure storage of private keys with hardware-assisted trusted environments, such as a trusted execution environment (TEE). Since a private key is strictly kept secret within the TEE and never leaves the device, the user must register the key separately for each combination of device and Web account, which is burdensome for users who want to switch devices. The aim of this research was to provide a solution for key management with enhanced usability by relaxing the restriction that keys can never leave the device and allowing private keys to be shared across devices while still maintaining an acceptable level of security. We propose a secure method for sharing keys across the TEEs of devices. The method has two functions: 1) trusted third party (TTP)-based device owner identification, which involves a TTP that is responsible for supervising key sharing across devices in an authentication system, and 2) secure key copy, which enables the duplication of keys in a device that were originally stored in another device through a direct secure transport channel between the TEEs of the devices. A TTP identifies the owner of each device to mitigate the risk of the keys being illegally shared. In this study, we evaluated the secure-key-copy function of our proposed method by implementing it in the ARM TrustZone-based TEE, showing that this function is feasible for commercially available smartphones.

Keywords: public key authentication, Fast IDentity Online (FIDO), key management, Trusted Execution Environment (TEE), Internet of Things (IoT)

1. Introduction

In modern Web applications, user authentication is essential for preventing unauthorized access, providing customized services, and other important functions. Many researchers have proposed various methods of replacing passwords [1], which have many problems in terms of both security and usability [2], [3]. One alternative to passwords is public key authentication, but the main obstacle to its successful adoption is the difficulty in protecting the private key from being stolen or misused [4]. Many personal mobile devices (e.g., smartphones) now offer a secure computing environment in which keys can be managed in a strictly secure manner with hardware-assistance such as trusted execution environments (TEEs) [5]. The Fast Identity Online (FIDO) Alliance defines a universal authentication framework (UAF) for password-less authentication based on public key cryptography that can be securely implemented on widespread TEE-enabled mobile devices [6].

FIDO-based authentication enables a simple and secure user experience by using a mobile device as a secure storage of private keys for authentication (authentication keys). However, since the authentication keys are tightly coupled to the device and generally cannot be migrated from one device to another, when a user replaces a device with a new one due to device lifecycle events (e.g., upgrading a phone) or begins using a new device such as a tablet or Internet-of-Things (IoT) device, the user must re-register the key on the new device for each Web account. As registration usually requires a certain manual action (e.g., typing a password, and/or sending a confirmation code via short message service (SMS)) by the user, such re-registration for each device and account is problematic from the perspective of user experience. We can improve user experience by securely sharing authentication keys among devices. This is important especially for services that currently protect users' accounts using password-based authentication methods and are frequently used in daily life. Examples are social networking services (SNSs) and e-commerce.

Considering device-lifecycle events, our goal was to provide a user with a solution with enhanced usability in key management for services described in the previous paragraph (e.g., SNSs and e-commerce). Our basic approach is to relax the restriction that authentication keys can never leave a device and allow keys to be shared across devices while mitigating the risks caused by key sharing and maintaining a certain level of security. In this paper, we propose a secure authentication-key-sharing method. This method has two functions: *trusted third party* (*TTP*)*-based device-owner identification* and *secure key copy*. With TTP-based device-owner identification, the owner of each device is identified by using a certificate issued by a TTP to prevent the keys from be-

[1] NTT Network Service Systems Laboratories, NTT Corporation, Musashino, Tokyo 180–8585, Japan
a) yoshihiko.omori.ak@hco.ntt.co.jp
b) takao.yamashita.cz@hco.ntt.co.jp

A preliminary version of this paper appears in the proceedings of the 4th International Conference on Mobile And Secure Services (MobiSecServ), 2018, doi: 10.1109/MOBISEC-SERV.2018.8311436.

ing illegally shared to another person's device. Secure key copy enables the duplication of keys in a device that were originally stored in another device. This function mitigates the risks of key theft during the sharing process between legitimate devices by establishing a secure communication channel between the TEEs of the devices. We evaluated the secure-key-copy function, showing that it is feasible to implement this function in the TEEs of commercially available smartphones. The feasibility of our proposed method had to be demonstrated by developing and running the method on a smartphone, because a TEE has a more limited development and execution environment than that provided by a common mobile operating system (OS) such as Android.

The rest of this paper is organized as follows. In Section 2, we elucidate a key problem with FIDO UAF and give an overview of the proposed method. In Sections 3 and 4, we describe the TTP-based device-owner-identification and secure-key-copy functions of our method, respectively. In Section 5, we discuss the evaluation results of the secure-key-copy function. We discuss the feasibility of this function from the point of view of implementing and running it on a commercially available smartphone. In Section 6, we compare our method with related methods. Finally, we conclude the paper in Section 7.

2. Usability Problem with FIDO UAF and Key-Sharing Solution

2.1 FIDO Universal Authentication Framework

The FIDO UAF [7] offers a password-less user experience in Web authentication. The FIDO authenticator, which is responsible for managing authentication keys, has three main functions, as illustrated in **Fig. 1**: *key management*, *local-user verification*, and *attestation*. The key-management function generates an authentication key-pair at registration time for each Web account and keeps the private key secret within the authenticator while registering the public key to a server. This function also responds to a cryptographic challenge from the server by making a signature with the private key, but the private key never leaves the authenticator even at this time.

In contrast, the local-user-verification function ensures that the user who registered the key is definitely using the key. Therefore, local-user verification is executed when the authentication key is first created and each time the key is used for making a signature. Biometric authentication based on certain physical characteristics (e.g., fingerprint, iris, or face) is widely used as a user-friendly method for local-user verification.

The attestation function provides a means to prove the capability of an authenticator. An authenticator is identified on the device model level by using an authenticator attestation identifier (AAID). The capability of an authenticator with an AAID is confirmed over a network by using an attestation key, which consists of an attestation certificate including a public key and corresponding private key. The private key of an attestation key is stored in an authenticator when it is manufactured. The capability information of an authenticator is provided by a server called a metadata server. We can also retrieve the attestation certificate of an authenticator from a metadata server by specifying its AAID. An authenticator proves its capability by sending a

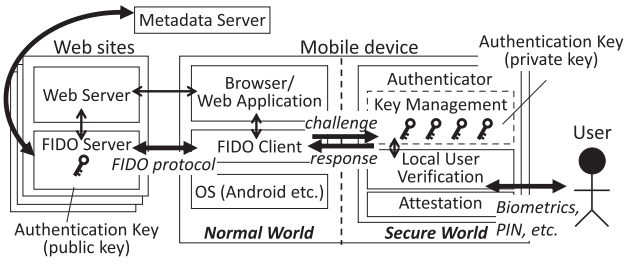

Fig. 1 Secure implementation of authenticator based on FIDO UAF.

message that includes its AAID and a signature generated by using the private key of its attestation key. When the public key of an authentication key pair is registered to a server, the AAID of an authenticator is also sent. At this time, a server determines that the capability of an authenticator satisfies service requirements. If it does not satisfy service requirements, the registration of a public key is rejected. Examples of the capability information of an authenticator are an authentication algorithm (e.g., ECDSA based on SECP256r1), key-protection method (e.g., TEE or SE), and the accuracy of a biometric authentication function [8], [9].

Thanks to support from trusted hardware, modern mobile devices have two execution domains: *normal world* and *secure world*. Most applications and system software including OSs, such as Android, run in the normal world. Although the execution environment in the normal world is rich and flexible, it is also susceptible to viruses and malware attacks. Therefore, applications that require higher levels of security, such as payments, digital rights management, and authentication, are implemented in the secure world, which is resistant to such attacks. The FIDO UAF was also designed to implement the authenticator in the secure world to enhance security [10].

2.2 Usability Problem due to Device-lifecycle Events

As described in the previous subsection, with the FIDO UAF, authentication keys are assumed to never leave the local device to ensure high security. Therefore, when a user starts using a new mobile device due to device-lifecycle events (e.g., upgrading to a new phone), authentication keys stored in the old device cannot be migrated to the new device. Thus, the user must re-register an authentication key newly generated on the new device for each Web account. To complete this re-registration, the user first needs to log-in to an account by using a non-FIDO fallback authentication method (e.g., password, secret question, or verification code sent by SMS/Email), which is usually less secure and user friendly [11]. Furthermore, the number of re-registrations is the same as the number of Web accounts on which users are registered. This is a serious usability issue in the real world because a survey in 2015 reported that 44% of Americans usually upgrade their smartphone every two years [12], and another report said that the average Web user has 27 accounts [13].

On the other hand, FIDO-supported devices and authentication servers have been widely deployed. In addition, the number of Web servers that provide commercial services and support FIDO-based authentication methods is increasing. Therefore, we have to solve the usability problem just described without changing the FIDO UAF protocol, which is used by a user device, a FIDO

server, and metadata server.

2.3 Proposed Method: Key Sharing across Devices

The motivation behind this research was to provide the user with a solution with enhanced usability. Our approach is to relax the restriction that keys can never leave devices and allow them to be shared across multiple devices to eliminate burdensome re-registration. We want to emphasize that our proposed method is not for freely allowing key migration between devices, which can significantly degrade high security achieved by the FIDO-based authentication method, but to enable secure key-sharing where an acceptable level of security for both users and service providers is maintained by mitigating the following risks potentially caused by key sharing:

(1) **Impersonation:** There is a risk that an attacker impersonates a legitimate user and steals keys from the user's device (*risk 1-A*). Furthermore, a user might transfer or share his/her account restricted for individual use with others (*risk 1-B*). This means that inter-user key sharing is forbidden whereas inter-device key sharing by a single user is permitted.

(2) **Keys stolen during delivery:** The other risk is that an attacker compromises the key-sharing process between legitimate devices and steals keys while they are being delivered from one device to another. We consider network attackers and malware attackers [14] as major security threats in key sharing. Network attackers try to steal keys by intercepting a communication channel between legitimate devices (*risk 2-A*). Malware attackers try to steal keys by running malicious software on legitimate devices (*risk 2-B*).

To mitigate the impersonation risks (*risks 1-A* and *1-B*), we introduce a TTP that is responsible for supervising key sharing across devices in an authentication system. A TTP identifies the owner of each device to mitigate the risks of sharing keys between the devices owned by different individuals. To protect keys from being stolen by a network attacker (*risk 2-A*), devices communicate with each other for sharing keys through a secure channel with encryption based on certificates issued by the TTP. Furthermore, we mitigate the risk of malware attacks (*risk 2-B*) by implementing a key copy function within the secure world. The detailed design of our proposed method and the procedure for copying keys securely from the secure world of one device to another are given in the following sections.

3. TTP-based Device-owner Identification

3.1 Problems with Pairing-based Verification

One important principle in the FIDO ecosystem is to ensure that only the user who registered the key can use it. This means that it must be verified with our key-sharing method with which authentication private keys are only shared between devices owned by the same individual.

Modern personal mobile devices (e.g., smartphones) are designed to verify that the pre-registered owner is definitely operating the device by means such as a pattern lock, personal identification number (PIN) lock, or biometrics. However, to restrict keys to be shared only between devices of the same owner, it is necessary to ensure that two locally verified owners are the same individual. A naïve pairing method involves asking the user to input the same secret authentication code to both devices after each owner is locally verified. This pairing method is the most convenient and prevalent for forming a trust relationship between devices. However, this naïve method works well only when the user pairs devices carefully, for example, by using a long code and avoiding pairing in public places [15]. However, there is a concern that the majority of Web users are careless and can be led by an expert attacker to unintentionally pair their devices with unknown devices, resulting in key theft (*risk 1-A*). Furthermore, the user can intentionally pair his/her device with another's, that is, there is no resistance to *risk 1-B*.

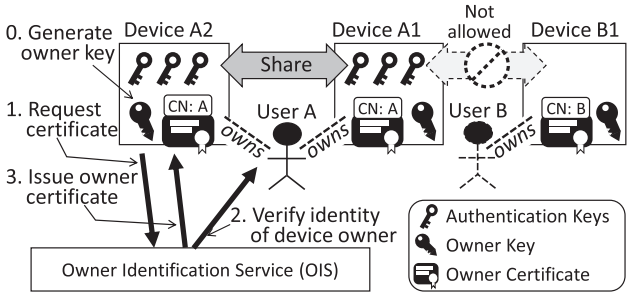

Fig. 2 Proposed method and procedure for issuing owner certificate.

3.2 Trusted Third Party Verifying Identity of Device Owners

To make the verification of device owners reliable enough for both users and service providers regardless of user behavior, we introduce a new entity called the Owner Identification Service (OIS), which supervises key sharing across devices as a TTP in an authentication system. As illustrated in **Fig. 2**, the OIS, acting as an authority of a public key infrastructure (PKI) system [16], ensures the identity of the owner of each device and issues a digital certificate (the owner certificate) to the device. The owner certificate accords with the common X.509 format, and the OIS sets two unique identifiers that are associated with the individual and his/her device in the distinguished name of a subject field defined for an X.509 certificate. An example of how to set these unique identifiers in a subject field is to set them as a common name (CN) in an X.509 certificate by concatenating them.

The private key related to the owner certificate (the owner key) should be protected in the secure world as securely as the authentication keys since the proof of possession of the owner key can trigger sharing of authentication keys. Therefore, the owner key is also protected in the secure world, and local-user verification is required each time the key is used.

After beginning to use a device, a user might upgrade it in a lifecycle event. In addition, a user device might be transferred to another user or lost. Therefore, an owner certificate installed in a user device has to be deactivated in such cases. To deactivate a PKI certificate, a certificate revocation list (CRL) [16] and/or an X.509 Internet Public Key Infrastructure Online Certificate Status Protocol (OCSP) [17] are used. When we use the OCSP, we can dynamically verify whether a PKI certificate is revoked at any time by accessing a server called an OCSP responder. If an owner certificate has been revoked, the authentication keys in a user de-

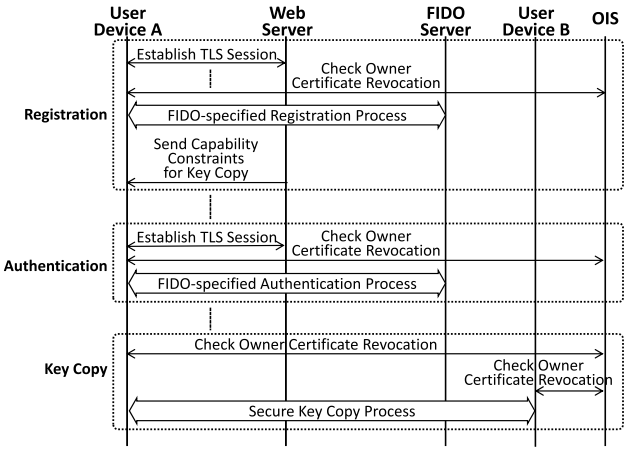

Fig. 3 Procedures for disabling authentication keys.

vice have to be disabled and deleted. **Figure 3** shows how the proposed method disables the authentication keys in a user device without changing the FIDO UAF protocol, which is needed for easy deployment, as described in Section 2.2. Before starting the FIDO UAF protocol, a TLS session is first established between a user device and Web server. A public key for user authentication is then registered to a FIDO server according to the specifications of the FIDO UAF protocol. To avoid the modification of the FIDO UAF protocol with the proposed method, a user device checks whether its owner certificate is revoked before the registration of a public key for user authentication. When an owner certificate has been revoked, the registration process of a public key is never performed until a user device is initialized. This owner certificate revocation check is also performed prior to executing the authentication process of the FIDO UAF protocol as shown in Fig. 3. We further explain Fig. 3 in Sections 4.2 and 4.3.

The OIS must offer a high level of identity assurance as the root of trust, which is quite costly due to the necessity of conducting many verification processes [18]. One example of a user verification process is an in-person verification process, where a user is verified by checking his/her identity (ID) cards in face-to-face communication. Another example is a verification process using an ID card with a smartcard function, which provides signature generation and verification using public-key cryptography. With the proposed method, we assume that the owner of a device is securely verified through such a verification process whenever he/she obtains a new device. Therefore, we believe it is reasonable for a business that already confirms people's identities in their normal business to also play the role of the OIS. In Japan, for example, mobile network operators are required by law to confirm the identity of a subscriber before selling him/her a phone and provide access to the mobile network. Users do not have to do any additional action for identity proofing if the mobile operator works as the OIS and issues the owner certificate at the time of contracting with users. The other most economical OIS candidate is a nationwide PKI system. Governments in some countries already provide the service of issuing certificates to mobile devices [19]. Also, businesses that already have contact points throughout the nation and usually confirm the identities of their customers (e.g., banks or postal services) are expected to cost-effectively act as the OIS.

4. Secure Key Copy across Devices

4.1 Peer-to-peer Key Copy between Devices

Key sharing across devices can be implemented using cloud storage where backup of authentication keys are stored in a cloud server and distributed to specific devices that have access privileges [20]. However, sharing secret data, such as authentication keys, using an Internet server increases the risk of scalable cyber-attacks by network attackers who want to steal keys from the server.

Our proposed method enables a user to securely copy authentication keys between devices in peer-to-peer fashion. Note that this secure-key-copy function should not export authentication keys in a form that enables any other individual to use them. It must be highly secure against attacks designed to steal keys by network attackers. Thus, devices create a direct transport channel between them using proximity communication technology, such as near field communication [21] or Bluetooth low energy [22], to make it difficult to attempt attacks remotely via the Internet. Furthermore, authentication keys are encrypted using the owner certificate and delivered from one device to another through the channel in an encrypted form to prevent eavesdropping attacks on the network.

4.2 TEE-based Secure Implementation of Key Copy

To make key copy also resistant to malware attacks, we implement the secure-key-copy function within the secure world of a device. **Figure 4** shows an overview of the secure implementation of this function. This function has two sub-functions required for sharing keys implemented as a software program running in the secure world. The secure world ensures that the internal software is isolated from other software and protected from any attack from malware usually running in the normal world. A TEE is available on many modern mobile devices as a secure world to which trusted applications can be flexibly added.

One required sub-function is a subscriber function (owner-verification sub-function in Fig. 4) of the PKI system provided by the OIS. This sub-function generates and stores the owner key, requests the OIS to issue the owner certificate, and makes a signature to prove the possession of the owner key. When a request for the issuance of an owner certificate is sent by this sub-function, it is signed with the owner key, as defined in PKCS#10 [23]. This request is further signed with the private key of an attestation key stored in an authenticator. The OIS can verify that a request for the issuance of an owner certificate is created by an authenticator in the secure world by verifying a signature generated using the private key of an attestation key. The other required sub-function is for creating a direct transport channel and copying keys between devices (copying sub-function in Fig. 4).

As described in Section 2.1, when a user device registers a public key for user authentication to a FIDO server, this server determines whether the capability of a user device's authenticator can satisfy service requirements. When an authentication key is copied from the authenticator of one device (sender device) to that

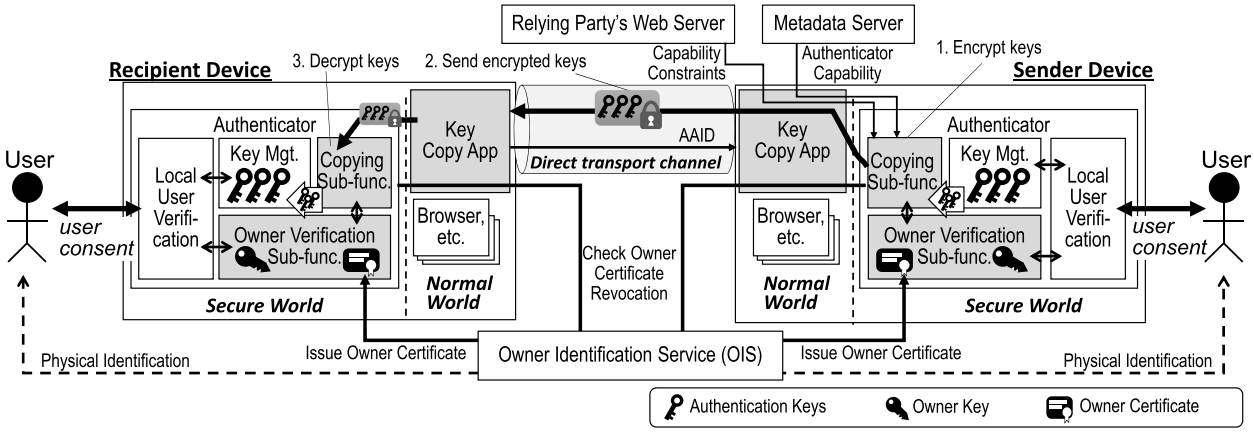

Fig. 4 Secure implementation of secure-key-copy function of our method based on owner-identification service.

of another (recipient device), the capability of a recipient device's authenticator must also satisfy service requirements. Therefore, with the proposed method, before sending an authentication key, the copying sub-function of the sender device determines whether the capability of the recipient device's authenticator satisfies service requirements as follows. After the sender device registers an authentication key to a FIDO server through a Web server, this Web server sends capability constrains for key copy back to the sender device, as shown in Fig. 3. These capability constraints contain conditions that the capability of the recipient device's authenticator must satisfy to receive an authentication key from the sender device. Before an authentication key is copied from the sender device to the recipient one, the recipient device sends its AAID to the sender device. The sender device then determines whether the capability of the recipient device's authenticator can satisfy the capability constrains corresponding to an authentication key. The TEE-based procedure for copying keys securely from one device to another is described in detail in the next subsection.

4.3 Procedure for Copying Keys from One Device to Another

Figure 5 illustrates the detailed procedure of copying keys from the TEE of one device (sender device) to that of another (recipient device) while confirming the devices have the owner certificate issued to the same individual (to mitigate *risks 1-A* and *1-B*) as well as encrypting keys during the delivery, as described in previous sections (to mitigate *risks 2-A* and *2-B*). The following steps are for copying keys. Prior to the following steps, the sender and recipient devices first check whether their owner certificates have been revoked as shown in Fig. 3.

(1) **Requesting copy:** The recipient device requests the sender device to copy keys by sending a message that includes the owner certificate and AAID of the recipient device. This message is signed with the private key of the attestation key of the recipient device. This signature represents that the recipient's owner certificate and its corresponding private key are managed in the secure world of the recipient device. The AAID in the message is used for verifying the signature associated with the message and determining whether the capability of the recipient device's authenticator satisfies the capability constraints associated with authentication keys. Local-user verification is required to verify that the owner of the recipient device is definitely requesting this action.

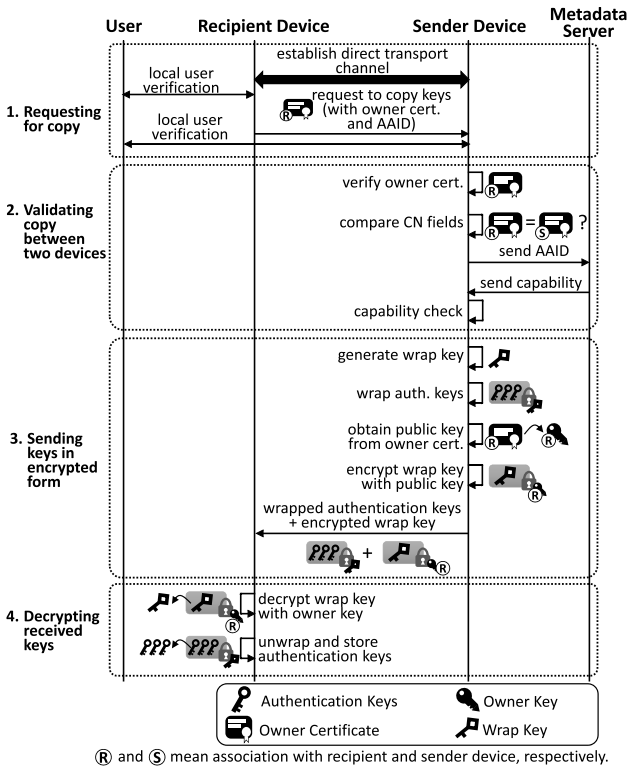

Fig. 5 Detailed procedure for copying keys securely between devices.

(2) **Validating copy between two devices:** The sender device validates before starting the actual copying process. First, the sender device verifies the recipient's owner certificate and the signature generated by using the attestation key of the recipient device. Through this signature verification, it can determine that the received message in step 1 is created by the authenticator of the recipient device. The sender device then compares the owner certificate of the recipient device with that of its own to confirm that the unique identifiers of individuals, which are the owners of user devices, in the CN fields of both certificates are the same. The sender de-

vice also requires local-user verification to ensure that the owner of the sender device agrees with copying the keys. The sender device then obtains the capability of the recipient device's authenticator from a metadata server using the AAID received from the recipient device and searches for authentication keys whose capability constraints are satisfied by the obtained capability.

(3) **Sending keys in encrypted form:** The authentication keys must be encrypted in the secure world of the sender device by using the pubic key attached to the owner certificate of the recipient device so that the authentication keys can be decrypted only in the secure world of the recipient device where the associated private key (the owner key) is present. To achieve better performance in the encryption process, the authentication keys are encrypted with a newly generated symmetric key (wrap key), which is encrypted with the public key attached to the received owner certificate. The sender device sends the wrapped authentication keys and encrypted wrap key back to the recipient device at the end of this process. The wrapped authentication keys and encrypted wrap key sent by the sender device can be decrypted only by the recipient device. This is because the wrap key can be retrieved only by the holder of the private key corresponding to the recipient's owner certificate, which is only the authenticator of the recipient device.

(4) **Decrypting received keys:** The recipient device decrypts the wrap key with its owner key (the private key associated with the public key attached to the owner certificate), then unwraps the authentication keys with the wrap key, and stores the authentication keys locally.

4.4 Key Recovery from Lost Device

When a user has his/her devices in hand, authentication keys in a device can be copied to another device by using the proposed method. On the other hand, a user might lose his/her device where authentication keys are stored but never shared to any other device. If a user lost such a device, he/she has to re-register new authentication keys using a non-FIDO fallback authentication method as described in Section 2.2. To reduce this risk, we assume the proposed method can be used as follows.

A user has one or more devices that are placed in a comparatively safe place such as a user's home and less likely to be moved from there. We call them *key-backup devices*. In addition, we call a place where key-backup devices are placed a *home place*. When a user registers a new authentication key using a device other than a key-backup one, he/she shares the key to his/her key-backup devices. To achieve this conveniently for a user, a user device has an application that manages authentication keys not shared to his/her key-backup devices. We call this application a *key-backup application*. When a key-backup application finds its owner's key-backup device to which new authentication keys have not been shared, it notifies a user to share them to a key-backup device. If a user frequently comes back to his/her home place, the probability that newly generated authentication keys will be lost can be reduced because we can recover authentication keys from a key-backup device.

5. Implementation and Evaluation

We evaluated the proposed method in terms of the feasibility of the secure-key-copy function, security and user convenience of device-owner identification, and probability of key loss. We first describe the implementation of the secure-key-copy function in a smartphone in Section 5.1 then discuss evaluation results in Section 5.2.

5.1 Secure-key-copy-function Implementation

Many commercial mobile devices provide the secure world on the basis of the TEE standardized by the GlobalPlatform [5]. As this TEE provides a more limited development and execution environment than the normal world, it is essential to determine whether the secure-key-copy function can be implemented on the basis of the primitive pre-defined standard APIs [24] to determine whether this function can be feasibly deployed in a commercial environment. We implemented the secure-key-copy function as a trusted application running in the ARM TrustZone-based TEE [25]. We call this function the *key-copy trusted application* (*TA*). The key-copy TA works as part of a FIDO authenticator, which enables the authentication keys to be copied across authenticators in accordance with the procedure described in the previous section. The key-copy TA has two simple APIs used for copying keys from one authenticator to another: *getKey()* and *storeKey()*. Calling *getKey()* at a sender device retrieves one specified key from the local authenticator and returns it in an encrypted form. *storeKey()* is called at the recipient device with the encrypted key as an argument, and the key is decrypted and stored in the local authenticator. In the key-copy TA, the wrap and owner keys are generated as a 256-bit AES key [26] and 2048-bit RSA key [27], respectively.

5.2 Evaluation Results

We implemented the key-copy TA on a commercially available smartphone, the Samsung Galaxy S8 (SM-G950FD), and confirmed that it works in the standard TEE on the device. We also measured the time required for copying keys to evaluate the feasibility of this function from the perspective of user experience. We measured the time for obtaining the number of (N) keys from the authenticator by calling *getKey()* and storing N keys in the authenticator by calling *storeKey()*. The N was varied from 1 to 40 by taking into account the survey results of Web site registration [13]. The time for sending the encrypted keys through the direct transport channel between the devices was excluded from the evaluation because we focused on the overhead caused by moving keys from/to the TEE in an encrypted form, which is a notable characteristic of our proposed method.

As shown in **Figs. 6** and **7**, the more authentication keys, the longer the required time, which is plotted with black circles in the figures. It took about 0.77 and 3.4 *s* to obtain and store 40 keys, respectively. Decryption computation with the private key (the owner key) for storing keys takes about five times longer than encryption computation with a public key for obtaining keys.

The breakdown of processing time for each API call is shown in **Table 1**. As described in the previous subsection, APIs

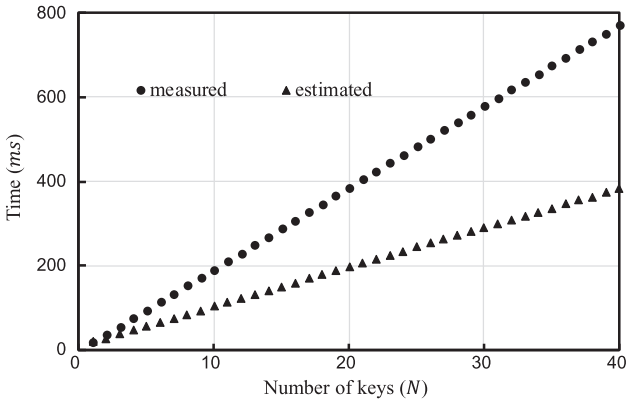

Fig. 6 Time for obtaining N encrypted keys from TEE.

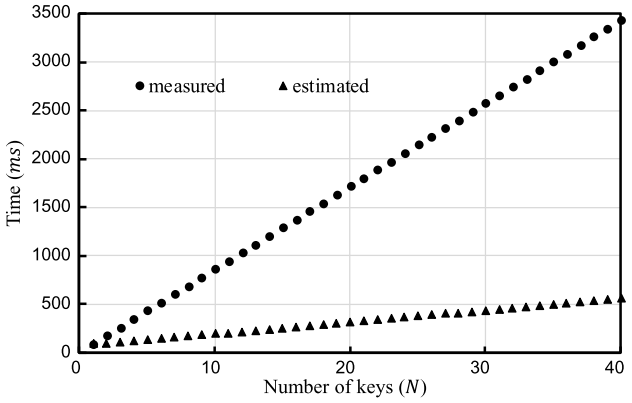

Fig. 7 Time for storing N encrypted keys after decryption in TEE.

Table 1 Breakdown of Processing Time for APIs.

API	Processing steps	Time (*) (*ms*)
a. *getKey()*	Step 1a: Generate wrap key (AES-256)	1.7
	Step 2a: Wrap authentication key with wrap key	9.3
	Step 3a: Encrypt wrap key with owner key (RSA-2048, public key)	8.3
b. *storeKey()*	Step 1b: Decrypt wrap key with owner key (RSA-2048, private key)	72.8
	Step 2b: Unwrap wrapped key with decrypted wrap key	13.1

(*) Average time of 60 API calls

getKey() and *storeKey()* implemented in the key-copy TA obtain and store one key for each API call, respectively. Therefore, when N keys are copied, the API is called N times; thus, all steps in Table 1 are repeated N times. However, it is possibly more efficient to provide extended APIs that obtain or store multiple keys with one API call. In this case, as one wrap key can be shared for copying N keys, steps 1a, 3a, and 1b in Table 1 need to be performed only once, whereas steps 2a and 2b need to be repeated N times. By using the extended APIs, 40 keys can be estimated to take 0.38 *s* to obtain and 0.56 *s* to store. These estimated times are also plotted with black triangles in Figs. 6 and 7. Therefore, the overhead for copying 40 keys between the TEEs of two devices is expected to be less than 1 *s*. As the average user has been reported to have fewer than 40 keys [13], we believe the 1-*s* overhead for copying keys is feasible from the perspective of user experience. Hence, these results indicate the feasibility of the secure-key-copy function of the proposed method on a commercially available smartphone.

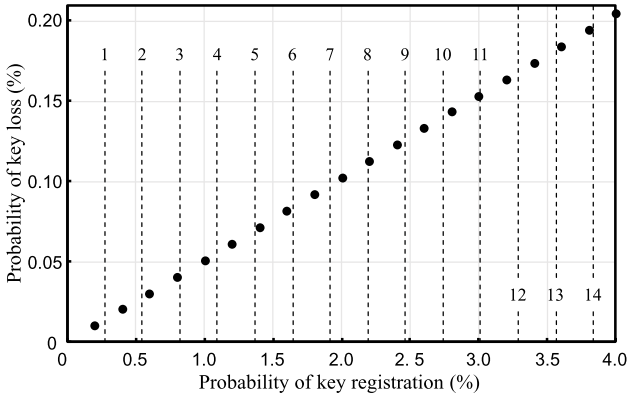

Fig. 8 Probability of key loss for one year.

When a user performs the copying of authentication keys between two devices, two steps are needed: unlocking a user device and verifying the equality of owners of two devices through a user identification process. As described in Section 3, the proposed method uses owner certificates installed in user devices in order to verify the equality of device owners. On the other hand, the existing method verifies this by validating two devices to be controlled by one user. For one example, a user is identified using a password and/or digits, which are used in iCloud keychain [20] and a method of pairing two devices with Bluetooth functionality. These identification methods require a user to input a password and/or digits to one or more devices although a user identification process is automatically performed from the second time in the case of iCloud keychain. This identification process is performed using the pair of a public key and private key generated in the first identification process. For another example, a user is required to place two devices back to back, which is used in Android Beam [28]. These methods require additional user operation for verifying the equality of device owners after unlocking devices. They can also be operated between two devices belonging to different owners. However, the proposed method does not require any additional operation for verifying the equality of device owners and cannot be used between two devices of different owners.

We finally discuss the probability of key loss when the proposed method is used as described in Section 4.4. For simple evaluation, we assume that authentication keys in a user device are shared to a key-backup device at a constant period (T_b). We call this period a *backup period*. Let P_d and P_r be the probabilities of occurrences of device loss and the registration of one authentication key for T_b, where we assume that one authentication key can be registered at most in T_b. We can then calculate the probability of authentication-key loss for a year P_k as

$$P_k = \{1 - (1 - P_d P_r)^{N_y}\}, \quad (1)$$

where N_y is the number of backup periods in a year. In this formula, we adopt P_d representing that 5% of devices are lost every year [29]. **Figure 8** shows the probability of authentication-key loss P_k, where T_b is 1 day. The horizontal and vertical axes are P_r and P_k, respectively. The vertical dashed lines indicate P_r at which the expectations of the number of authentication keys newly registered in a year range from 1 to 14. For example, the

dashed line below "7" in the graph indicates P_r at which the expectation of the number of authentication keys newly registered in a year is 7. From this figure, the probability of key loss is around 0.21% at most. This probability means that a user will never lose keys for 100 years with the probability of more than 80%. Hence, we can effectively reduce the probability of key loss, which requires a user to register new keys using non-FIDO fallback authentication methods.

6. Related Work

Exporting authentication private keys has been traditionally considered convenient for restoring authentication keys on a different device [30]. However, as public key authentication becomes widely used to protect accounts with higher security requirements, such export may degrade security; therefore, a standard hardware-assisted key management application program interface (API) provided by a modern mobile OS [31] does not allow private keys to be exported. With the FIDO UAF, it is also assumed that authentication private keys never leave the local device for security reasons [32], causing the usability problem of re-registration when switching devices. To improve the usability of FIDO-based authentication without critically compromising the high security against theft or misuse of authentication private keys, our proposed method allows the sharing of keys across devices based on the device-owner's identity instead of allowing the export of authentication keys in a form that enables any other individual to use them.

There are methods that can be used for mitigating the burden of re-registration for each Web account. Identity-management methods, such as OpenID and a Security Assertion Markup Language (SAML) [33], enable a single sign-on across multiple Web accounts for reducing the number of authentications. The FIDO UAF is expected to be an initial stronger authentication method at an identity provider [7]. In such a federated use case, the user has to register the new device again, not to all Web accounts but only to the identity provider. However, not all Web accounts will rely on a single identity provider [34]; thus, the user still has to complete the registration for all identity providers the user is using and for Web accounts that do not rely on any identity providers. The Client-to-Authenticator-Protocol (CTAP) [35] enables multiple client devices (PC, phones, etc.) to remotely access the authentication keys located in the external authenticator. In this use case, the user does not have to register each client device but still has to re-register for each account after replacing the external authenticator with a new one. Although the methods mentioned above may mitigate the burden on users at a certain level, re-registration is still needed in some situations. Our proposed method completely eliminates the necessity of re-registration by enabling secure migration of keys.

There is also a method for mitigating the burden of registration for each Web account by making it transparent to the user. The transfer access protocol [36] is an extension of the FIDO UAF for registering public authentication keys generated in a new device. It requires the creation of a chain of trust between old and new devices. The trust chain of an old and new device is created using the authentication key of an old device. The public key created by a new device is associated with the already registered public key of an old device using a signature calculated with the authentication key of the old device. Then, a new device can access a server using the authentication key newly created in it. Assume that a user has multiple devices that contain their own authentication keys to use a service and has lost one of the devices at a particular time. We can delete only a public key, which corresponds to an authentication key in the lost device, in a FIDO server to avoid improper use of a user account because different authentication keys are used for multiple devices. When using the Transfer Access Protocol, we can thus control the access of a user to a server on the side of a FIDO server. On the other hand, it is difficult for common users to inform a service provider about a public key included in a lost device. Therefore, we have to further improve usability when using this protocol. From the viewpoint of deployment, we have to revise the specifications of the FIDO UAF protocol to use the Transfer Access Protocol. This means that all user devices and FIDO-related servers including those already deployed have to support a new specification including the function of the Transfer Access Protocol. In contrast, the multiple devices of a user share the same authentication key to use a service with our proposed method. It can disable the authentication keys in a lost device by notifying an OIS about the event of its loss as described in Sections 3.2 and 4.3. The proposed method also does not require any revision to the specifications of the FIDO UAF protocol. Therefore, it is unnecessary to update the functions of user devices and FIDO-related servers that do not need the proposed method. This is important for the deployment of the proposed method.

7. Conclusion

We proposed a method of allowing authentication keys to be shared across personal mobile devices as a solution to the usability issue of Fast Identity Online (FIDO)-based authentication caused by device-lifecycle events. This method is also suitable for Internet of Things (IoT) devices. Our motivation was to provide the user with a solution that offers enhanced usability while mitigating the risks caused by key sharing and still maintaining an acceptable level of security. Our method has a trusted third party (TTP)-based device-owner authentication function. With this function, we introduced a TTP that is responsible for supervising key sharing across devices in an authentication system. The TTP identifies the owner of each device to prevent the authentication keys from being shared with a device of a different individual. Our method also has a secure-key-copy function for copying the keys securely from one device to another, which mitigates the risk of the keys being stolen by attackers during key-sharing. This function creates a direct transport channel between devices, and the keys are delivered through the channel in an encrypted form. Encryption is based on a user certificate issued to each device by the TTP and ensures that the keys are securely copied from the trusted execution environment (TEE) of one device to that of another. We implemented the secure-key-copy function running in the standard ARM TrustZone-based TEE on a widely used smartphone and evaluated it. The evaluation showed that several keys (the number of accounts an average user is ex-

pected to have) are expected to be copied in less than 1 *s*. This result indicates the feasibility of this function on commercially available mobile devices.

To demonstrate the fundamental feasibility of our proposed method, we have implemented a core part of the proposed method enabling keys to be securely delivered between two different secure worlds. For real-world deployment, the full key-sharing method needs to be implemented to evaluate the end-to-end user experience with the proposed method.

References

[1] Bonneau, J., Herley, C., van Oorschot, P.C. and Stajano, F.: The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes, *Proc. 2012 IEEE Symposium on Security and Privacy*, pp.553–567 (2012).

[2] Morris, R. and Thompson, K.: Password Security: A Case History, *Comm. ACM*, Vol.22, No.11, pp.594–597 (1979).

[3] Adams, A. and Sasse, M.A.: Users Are Not The Enemy, *Comm. ACM*, Vol.42, No.12, pp.41–46 (1999).

[4] Kleppmann, M. and Irwin, C.: Strengthening Public Key Authentication against Key Theft, *Proc. 9th Int'l Conf. Passwords*, pp.144–150 (2015).

[5] GlobalPlatform: TEE System Architecture Version 1.2, GPD_SPE_009 (2018).

[6] GlobalPlatform: Practical Business Considerations: Realizing FIDO Authentication Solutions with GlobalPlatform Technologies, White Paper (2018), available from ⟨https://globalplatform.org/resource-publication/practical-business-considerations-realizing-fido-authentication-solutions-with-globalplatform-technologies/⟩.

[7] Machani, S., Philpott, R., Srinivas, S., Kemp, J. and Hodges, J. (Eds.): FIDO UAF Architectural Overview, FIDO Alliance Proposed Standard 08 (2014), available from ⟨https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-uaf-overview-v1.0-ps-20141208.pdf⟩.

[8] Lindemann, R. and Kemp, J.: FIDO Metadata Statement, The FIDO Alliance (2017), available from ⟨https://fidoalliance.org/specs/fido-uaf-v1.1-ps-20170202/fido-metadata-statement-v1.1-ps-20170202.html⟩.

[9] Lindemann, R.: FIDO Registry of Predefined Values, The FIDO Alliance (2018), available from ⟨https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-registry-v2.0-id-20180227.html⟩.

[10] Lindemann, R. (Ed.): FIDO Authenticator Security Requirements, The FIDO Alliance (2018), available from ⟨https://fidoalliance.org/specs/fido-security-requirements-v1.2-2018/fido-authenticator-security-requirements-v1.0-wd-20180629.html⟩.

[11] Javed, A., Bletgen, D., Kohlar, F., Dürmuth, M. and Schwenk, J.: Secure Fallback Authentication and the Trusted Friend Attack, *Proc. IEEE 34th Int'l Conf. Distributed Computing Systems Workshops*, pp.22–28 (2014).

[12] Swift, A.: Americans Split on How Often They Upgrade Their Smartphones (2015), Gallup, available from ⟨http://www.gallup.com/poll/184043/americans-split-often-upgradesmartphones.aspx⟩.

[13] Stobert, E. and Biddle, R.: The Password Life Cycle: User Behaviour in Managing Passwords, *Proc. 2014 Symposium on Usable Privacy and Security*, pp.243–255 (2014).

[14] Lang, J., Czeskis, A., Balfanz, D., Schilder, M. and Srinivas, S.: Security Keys: Practical Cryptographic Second Factors for the Modern Web, *Proc. 20th Int'l Conf. Financial Cryptography and Data Security*, pp.422–440 (2016).

[15] Gehrmann, C.: Bluetooth Security White Paper, Technical Report, Bluetooth SIG Security Expert Group (2002).

[16] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R. and Polk, W.: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, RFC5280 (2008).

[17] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S. and Adams, C.: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP, RFC 6960 (2013).

[18] Grassi, P.A., Garcia, M.E. and Fenton, J.L.: Digital Identity Guidelines, National Institute of Standards and Technology Special Publication 800-63-3 (2017).

[19] Gemalto: National Mobile ID schemes Volume I, White Paper (2014), available from ⟨http://www.gemalto.com/govt/coesys/mobile-id⟩.

[20] Apple, Inc.: iOS Security Guide, White Paper (2018), available from ⟨https://www.apple.com/business/site/docs/iOS_Security_Guide.pdf⟩.

[21] International Organization for Standardization: ISO/IEC JTC 1, available from ⟨https://www.iso.org/isoiec-jtc-1.html⟩.

[22] Bluetooth Technology, available from ⟨https://www.bluetooth.com/bluetooth-technology/radio-versions⟩.

[23] Nystrom, M. and Kaliski, B.: PKCS#10: Certificate Request Syntax Specification Version 1.7, IETF RFC 2986 (2000), available from ⟨https://tools.ietf.org/html/rfc2986⟩.

[24] GlobalPlatform Technology: TEE Internal Core API Specification Version 1.2, GPD_SPE_010 (2019).

[25] ARM: ARM Security Technology: Building a Secure System using TrustZone Technology, White Paper, PRD29-GENC-009492C (2009).

[26] National Institute of Standards and Technology: Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197 (2001).

[27] Moriarty, K. (Ed.), Kaliski, B., Jonsson, J. and Rusch, A.: PKCS #1: RSA Cryptography Specifications Version 2.2, RFC8017 (2016).

[28] Android Developers, available from ⟨https://developer.android.com/guide/topics/connectivity/nfc/nfc⟩.

[29] McAfee, LLC: Almost 5% of smartphones Lost Every Year, available from ⟨https://securingtomorrow.mcafee.com/consumer/family-safety/almost-5-of-smartphones-lost-every-year/⟩.

[30] Barker, E.: Recommendation for Key Management, Part 1: General, National Institute of Standards and Technology Special Publication 800-57 Part 1 Rev. 4 (2016).

[31] Android Open Source Project: Hardware-backed Keystore, available from ⟨https://source.android.com/security/keystore⟩.

[32] Lindemann, R. (Ed.): FIDO Security Reference, FIDO Alliance Implementation Draft (2018), available from ⟨https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-security-ref-v2.0-id-20180227.html⟩.

[33] Chadwick, D.W.: Federated Identity Management, *Foundations of Security Analysis and Design V*, Lecture Notes in Computer Science, Vol.5705, pp.96–120, Springer (2009).

[34] Loutfi, I. and Jøsang, A.: 1, 2, Pause: Let's Start by Meaningfully Navigating the Current Online Authentication Solutions Space, *Proc. 9th IFIP WG 11.11 International Conference on Trust Management*, pp.165–176 (2015).

[35] Brand, C., Czeskis, A., Ehrensvärd, J., Jones, M.B., Kumar, A., Lindemann, R., Powers, A., Verrept, J. (Eds.): Client To Authenticator Protocol (CTAP), FIDO Alliance Implementation Draft (2018), available from ⟨https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-client-to-authenticator-protocol-v2.0-id-20180227.html⟩.

[36] Takakuwa, A., Kohno, T. and Czeskis, A.: The Transfer Access Protocol - Moving to New Authenticators in the FIDO Ecosystem, Technical Report UW-CSE-17-06-01, The University of Washington (2017).

Hideo Nishimura received his B.E. degree in computer science from the University of Electro-Communications in 2005 and his M.S. degree in mathematical and computing sciences from Tokyo Institute of Technology in 2007. In 2007, he joined NTT, where he has been researching innovative technologies for a telecommunication core network and modern user authentication. He is a member of IEICE.

Yoshihiko Omori received his M.E. degree in electrical communication engineering from Tohoku University in 1993. Since joining NTT in 1993, he has been engaged in research on traffic control in IP-based networks, QoS control, operation systems for VPNs, authentication technologies, and packet networks for mobile communications at NTT Telecommunication Networks Laboratories and NTT DOCOMO. He is a member of IEICE.

Takao Yamashita received his B.S. and M.S. degrees in electrical engineering in 1990 and 1992 from Kyoto University, where he also received his Ph.D. degree in informatics in 2006. In 1992, he joined NTT. His current research interests encompass network security, cloud computing, Internet-of-Things, and distributed algorithms. He is a member of IEICE, IEEE, IPSJ, and APS.