Certificate Verification under FIDO Authentication

Momoko Shiraishi[1] Hitoshi Aida[1]

Abstract: As a variety of financial applications are offered, the security in the authentication of users or transactions is required. FIDO authentication is considered to be resistant to man-in-the-middle attacks in user authentication because only the signed authentication result is returned to the authentication server without sending any secret information. Accordingly, it enables authentication without passwords, which is more user-friendly and has recently been introduced into various applications. However, under the current authentication protocol, if any of the software modules comprising FIDO authentication is infected with malware and behaves improperly, it is possible to lead mis-binding attack, parallel session attack, or DoS attack. In this paper, we specify the attacking paths of which types are the mis-binding attack and the parallel session attack. Afterwards, we propose a protocol to authenticate each software module that constitutes FIDO authentication on a session-by-session basis in order to deal with these attacks.

Keywords: FIDO authentication, digital certificate, transaction authentication

1. Introduction

With a variety of financial applications being offered these days, user authentication systems are indispensable. FIDO (Fast IDentity Online) is an authentication method based on a public key infrastructure that prevents man-in-the-middle attacks by keeping authentication information out of the network. Moreover, it replaces conventional password management with biometrics and other lighter authentication methods, thereby providing a more user-friendly environment.

The protocol of FIDO is released by the FIDO alliance, a non-profit standards organization launched in July 2012. FIDO 1.0 consists of two technical specifications, UAF (Universal Authentication Framework) and U2F (Universal 2nd Factor). UAF was released in December 2014. Additionally, U2F was released in May 2015, which aims to be implemented as a second factor authentication using USB key, Bluetooth and NFC. FIDO 2.0, on the other hand, which was launched in November 2015, integrates both UAF and U2F. Since the World Wide Web Consortium (W3C) adopted a new approach to web authentication based on the FIDO 2.0 web API in February 2016, companies have started to implement the FIDO authentication methods on their web platforms. This resulted in FIDO being actively adopted in various businesses. However, not much specific research on the protocol has been reported. A recent study in 2021[5] firstly presented a detailed cryptographic formulation of the protocol. Although potential attacks were pointed out as limitations, no detailed countermeasures against such attacks have been provided.

In FIDO authentication, the communication between the user's device and the application server is secured by TLS communication. However, the authentication process within the device does not authenticate each of the multiple modules involved, resulting in potential malware infection. This paper provides a new scheme that each software provider is required to issue a digital certificate in their code, and users verify the certificate when they conduct FIDO authentication process. In the existing framework, iOS applications and Android applications already adopted regulations concerning the issuing of digital certificates to application developers. While sustaining the above scheme, as an extension, we suggest a framework to issue certificates for software modules dedicated FIDO authentication that are not yet covered.

The contributions of this paper are summarized as follows:

- We summarize the vulnerabilities of the FIDO authentication protocol, describing the new attacking paths related to the mis-binding attack and the parallel session attack. Those attacks are caused by malware infection of each software module.
- We provide the countermeasure against the aforementioned attacks. By requiring each software module to issue a digital certificate, the user surely detects the malware infection in the authentication process. Basically, the FIDO alliance or some other organization is assumed to be responsible for issuing the digital certificate.

The next section provides the conventional FIDO authentication scheme offered by FIDO alliance. Section 3 then summarizes the vulnerabilities under the conventional FIDO protocol. Section 4 proposes countermeasures solving the vulnerability issues. Lastly section 5 provides a conclusion.

2. Conventional FIDO Scheme

In this section, we describe the conventional FIDO authentication mechanism. There are two stages: the registration stage and the login authentication stage. First, the software components involved in these stages will be described. Then, the authentication process will be presented.

[1] The University of Tokyo, Japan

2.1 Components of FIDO Authentication Scheme

Since FIDO authentication is a mechanism for user authentication in application use, it is roughly a two-party communication structure between the application provider (Relying Party in Figure 1) and the user device as Figure 1 shows. Authentication in the user device is divided into UAF Agent, UAF client, ASM, and an authenticator. Mention that applications can be distinguished by the way they are obtained. The classification is as follows.

- Native application: Obtained from stores managed by OS vendors. Examples are Android app, iOS app, and Windows app.
- Web application: Obtained from the developer's website.

2.1.1 RP (Relying Party)

Refers to the provider of the application. The application provider needs to prepare a web server that communicates the actual service contents and a FIDO server that performs user authentication in accordance with the FIDO authentication mechanism. In addition, the application provider uses the meta database only in the initial process. In addition, the application provider uses the meta database only at the beginning of the process, which is a database containing digital key certificates corresponding to the pre-installed attestation keys in the user devices. This database needs to be consulted only for initial user authentication. *AppID* identifies the application provided each RP.

2.1.2 User Agent

Refers to the browser, which provides the results from the web app server to the client side. Chrome, Safari, and Firefox are typical examples. TLS communication is used between the RP and the User Agent. The *FacetID* identifies each Use Agent.

2.1.3 UAF Client

A software module dedicated for FIDO authentication to communicate with FIDO server. In the case of native applications, it is included in the application module itself. For web applications, on the other hand, it is a separate software module that is required. Each UAF client is distinguished by the index *CallerID*.

2.1.4 ASM (Authenticator Specific Module)

A software module that mediates between the UAF client and the authenticator. A token is issued at the beginning of use.

2.1.5 Authenticator

In a broad sense, it is a mechanism for verifying the authentication response of a user, while it consists of two components: the module that verifies the actual user's authentication response and executes the application authentication process, and the database that stores the keys used for authentication. In this paper, the part that performs the former function is defined as authenticator, while the latter is referred to as secure storage. In addition, the hardware device that actually catches the user's authentication information is denoted as a biometric reader. Note that there are two types of the authentication devices: those with built-in authentication devices such as biometric authentication or PIN code input, and those with external authentication devices such as USB. The former is called bounding authentication and the latter is called roaming authentication. Each authenticator module is distinguished by the index *AAID*.

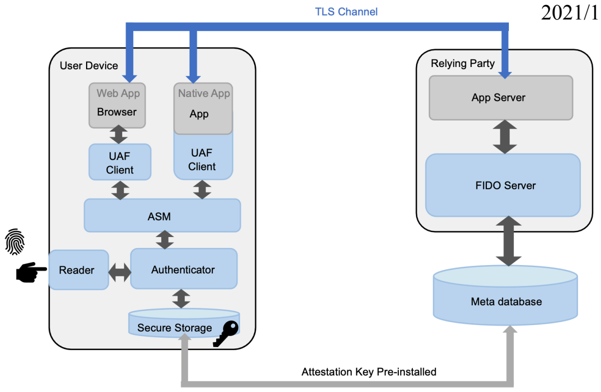

Fig. 1 Overview of the Authentication Scheme

2.2 FIDO Authentication Procedure

FIDO authentication consists of two phases: A) public key registration phase as described in Figure 5 and B) actual login authentication phase as described in Figure 6. Table 1 describes the notation of each parameter in the protocol, including those in the proposed protocol described below in section 4. While the protocol has been formulated in previous studies[5], we provide a more precise description here, referring to a report published by the FIDO alliance[1][12] [2]. The following subsections specify the individual phases including the set up.

2.2.1 Set up

As a prerequisite, before the user actually registers the application initially, the user's device is pre-distributed with a secret key called the attestation key sk_{AT}. This is stored in the secure storage. The public key pk_{AT} corresponding to the secret key sk_{AT} is stored in a meta database that can be accessed by the application provider. This mechanism is collectively referred to as the meta database service.

2.2.2 Registration Phase

When the user clicks on the URL of the web application, the user agent receives the click response from the user and accesses the website, based on TLS communication. The application server sends the username, $AppID$, and challenge to the UAF client. The UAF client calculates the challenge parameters fcp after verifying $FacetID$ and sends them to the ASM. The ASM then calculates ak and $hash(fcp)$, and sends them to the authenticator. The authenticator generates a pair of private and public authentication key $(sk_{AU}\ \ pk_{AU})$ and sends them back to the ASM. The ASM receives them and returns them to the application server. The application server sends them to the FIDO server, which verifies the signature. If the verification is successful, the authentication public key pk_{AU} is stored.

2.2.3 Authentication Phase

When the user clicks on the URL of the web application, the user agent receives an action from the user and accesses the website. The application server sends the $KeyID$ which corresponds to the username, $AppID$, challenge, and transaction text Tr to the UAF client. The UAF client verifies $FacetID$ and calculates the challenge parameters fcp, and send them to the ASM. The ASM calculates ak and $hash(fcp)$, and send them to the authenticator. The authenticator verifies ak at first. After obtaining the user's confirmation, the authenticator increments the counter $CNTR_A$. After signing the information including the transaction text Tr, the authenticator sends them to the ASM. The ASM, the UAF client and the user agent receive them and return them to the application server. The application server sends it to the FIDO server, which verifies the signature. If the verification is successful, the confirmation result is sent to the user's device.

2.3 Conventional Entities

This section provides the actual identity figures that service providers are currently offering for their products. Table 2 shows what denotes the ID for each software module and who determines it. In particular, for $AppID$, $FacetID$ and $CallerID$, the identifiers are set by service providers, implying that they are set under the control of the app store provider for native applications: iOS apps, Android apps, and Windows apps. On the other hand, for web applications $AppID$ and $FacetID$ are identical to each HTTPS URL. For the other software components are managed depending on service and device providers.

3. Vulnerabilities

In the conventional FIDO process, the entity authentication between the PR server and the UAF client is proceeded with TLS communication. However, not all other communication within the device is authenticated in each transaction. Therefore, if the authenticator, ASM, and UAF client are infected with malware, authentication will not always be performed correctly. Here, to be infected with malware means to have a module that behaves in an unappropriate manner between modules. In this section, we will discuss what kind of attacks are possible based on the malware infection of the software modules that are not authenticated by the conventional protocols.

3.1 Authenticator Mis-binding Attack in Registration Phase

In the registration phase, the attackers try to register their own device, indicating that the attackers' authentication public key is registered to the application provider. The lurking place for the attacker is the authenticator itself or the ASM or UAF client. Specifically, if one of those software modules sends a request to the attacker's own authenticator before the registration request reaches the legitimate authenticator, the attack succeeds.

3.2 Pararell Session Attack in Authentication Phase

A parallel session attack is an attack in which a malicious software module exists between legitimate modules, and it sends a request again and obtains random values generated for each session, resulting in the success of the authentication process. The purpose of this attack can be simply to successfully authenticate the login or to establish a false transaction. The former case indicates that the legitimate user reacts and inputs the credential while the internal software module is malicious. If the attacker's goal is to browse or operate the application after the user has authenticated, the attacker also needs to interfere with systems other than the FIDO authentication module. The protocol mechanism of the attack is provided in [5], while this paper provides the attack of the latter case which requires the operation to falsify the transaction text. Again, this paper focuses the parallel session attack whose purpose is not login authentication, but authentication for transactions. In this case, the transaction text to be recognized by the user and ones that the attacker tries to complete are different, indicating that the attacker has to break through the process of the user looking at the display to confirm. Hence, except for the authentiactor malware infection, we assume a clickjacking attack at the same time. For the attack technique, we assume that the technique has been successfully implemented as in the previous works [6] [8] [10]. We divide the cases according to whether the biometric reader is inside or outside the device.

3.2.1 Bounding Authentication

Under bounding authenticating, we first consider the case where a malicious file exists between the legitimate ASM and the legitimate authenticator as described in Figure 2. Under the conventional scheme, the authentication of the ASM is based on whether the *ak* stored by the authenticator at the registration stage matches the *ak* sent by the ASM at each session. Therefore, although the ASM itself cannot generate the correct *ak*, once the *h* and *ak* are intercepted in communication with the authenticator, a false transaction can be established.

Next, in addition to the condition that a malware exists between the legitimate ASM and the legitimate authenticator, for the case of malware infection between the legitimate UAF client and the legitimate ASM, the validity of *CallerID* of the UAF client is not checked under the conventional method. Therefore, a malicious module exists between them successfully creates a fake transaction Tr_A, and sends it to the subsequent ASM. Afterwards, the user's signature on Tr_A can be obtained and its authentication will be completed.

3.2.2 Roaming Authentication

While the above was for bounding authenticating, now consider the case of roaming authenticating. First of all, the authentication phase of roaming authenticating is not tied to the value of the registration phase, and only the $AppID$ is used in each session, therefore the attacker only requests again and complete the fake transaction, without acquiring *h* or *ak*. For the case of malware between the legitimate authenticator and ASM as described in Figure 3, the attacker only needs to make the false transaction request Tr_A and sends it to the subsequent legitimate authenticator. Moreover, under the roaming case, regardless of whether a malicious module exists before the legitimate authenticator, a malware between the legitimate UAF client and the legitimate ASM succeeds to make a false transaction Tr_A and to obtain the user's signature as described in Figure 4.

4. Proposed Scheme

As a countermeasure to the above attacks, we propose a method of performing individual authentication for each software module. The predecessor/successor module in the communication constantly verifies the digital certificate of the successor/previous module file in the registration/authentication phase. Under the existing framework, the connection eventually reach the RP server whose trust is established through mutual authentication based on the TLS communication. Thus, if any module, single or multiple, is infected with malware, an anomaly can be detected at some point. The following subsections specify the protocol for each phase. Note that in the proposed method, the identical protocol is applied for the bound authenticator and the

Table 1 Parameter Description both in the Conventional Protocol and Proposed Protocol

<table>
<tr><th colspan="2">Notation</th><th rowspan="2">Description</th></tr>
<tr><th>Conventional Scheme</th><th>Proposed Scheme</th></tr>
<tr><td colspan="2">$AppID$</td><td>Identifier for each application</td></tr>
<tr><td colspan="2">$FacetID$</td><td>Identifier for each browser</td></tr>
<tr><td colspan="2">$CallerID$</td><td>Identifier for each UAF client</td></tr>
<tr><td>tok</td><td>-</td><td>Token which is gernerated in ASM in the initial step</td></tr>
<tr><td>-</td><td>$ASMID$</td><td>Identifier and also certificate for each ASM</td></tr>
<tr><td>-</td><td>$AAID_C$</td><td>Identifier and also certificate for each authenticator</td></tr>
<tr><td colspan="2">$AAID$</td><td>Identifier for each authenticator software module</td></tr>
<tr><td colspan="2">$UName$</td><td>Username, identifier a user's account</td></tr>
<tr><td colspan="2">$KeyID$</td><td>Identifier for each user replaced by $UName$ used in the registration phase</td></tr>
<tr><td colspan="2">sk_{AT}</td><td>Attestation secret key which is pre-installed in the user's device</td></tr>
<tr><td colspan="2">pk_{AT}</td><td>Attestation public key which is stored in the meta-database and corresponds to sk_{AT}</td></tr>
<tr><td colspan="2">sk_{AU}</td><td>Secret key for authentication</td></tr>
<tr><td colspan="2">pk_{AU}</td><td>Public key for authentication</td></tr>
<tr><td colspan="2">k</td><td>A symmetric key for eyncryption and decryption for confidential information</td></tr>
<tr><td colspan="2">$Chlg$</td><td>Random value for each session created by the FIDO server</td></tr>
<tr><td colspan="2">$SData$</td><td>Random value for each session created by the Relying Party</td></tr>
<tr><td colspan="2">$TLSData$</td><td>Identifier of TLS channel for each session</td></tr>
<tr><td colspan="2">fc</td><td>Final challenge for Challenge-Response mechanism</td></tr>
<tr><td colspan="2">$CNTR$</td><td>Counter for authentication sessions</td></tr>
<tr><td colspan="2">n</td><td>Random value which authenticator generates</td></tr>
<tr><td colspan="2">$[Tr]$</td><td>Transaction text</td></tr>
<tr><td>ak</td><td>-</td><td>Value for authenticating ASM</td></tr>
<tr><td>h</td><td>-</td><td>A key container generated in the registration phase</td></tr>
</table>

Table 2 IDs Classification

<table>
<tr><th colspan="2">Application Type</th><th>RP
($AppID$)</th><th>User Agent
($FacetID$)</th><th>UAF client
($CallerID$)</th><th>ASM</th><th>Authenticator
($AAID$)</th></tr>
<tr><td rowspan="2">Native application</td><td>iOS</td><td rowspan="2">App ID</td><td colspan="2">Bundle ID (issued by Apple)</td><td rowspan="3">Token is issued by application providers</td><td rowspan="3">$AAID$ is issued by application providers</td></tr>
<tr><td>Android</td><td colspan="2">APK signing certificate (issued by users themselves or Google)</td></tr>
<tr><td colspan="2">Web application</td><td colspan="2">https://fido.example.com</td><td>-</td></tr>
</table>

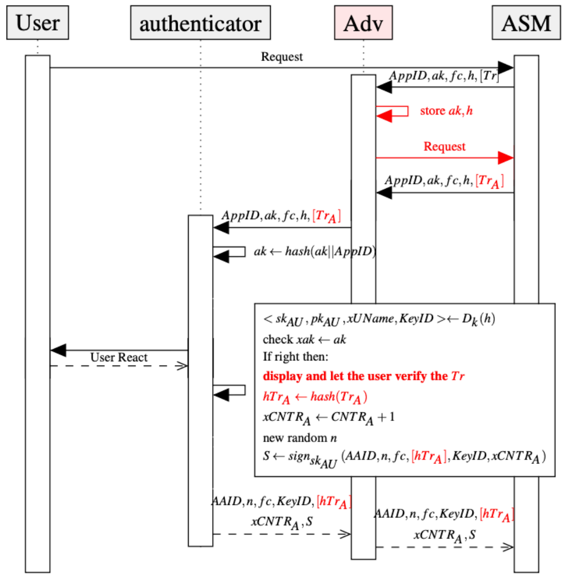
User
authenticator
Adv
ASM
Request
$AppID, ak, fc, h, [Tr]$
store ak, h
Request
$AppID, ak, fc, h, [Tr_A]$
$AppID, ak, fc, h, [Tr_A]$
$ak \leftarrow hash(ak||AppID)$
$< sk_{AU}, pk_{AU}, xUName, KeyID > \leftarrow D_k(h)$
check $xak \leftarrow ak$
If right then:
display and let the user verify the Tr
$hTr_A \leftarrow hash(Tr_A)$
$xCNTR_A \leftarrow CNTR_A + 1$
new random n
$S \leftarrow sign_{sk_{AU}}(AAID, n, fc, [hTr_A], KeyID, xCNTR_A)$
User React
$AAID, n, fc, KeyID, [hTr_A]$
$xCNTR_A, S$
$AAID, n, fc, KeyID, [hTr_A]$
$xCNTR_A, S$

Fig. 2 Malware between ASM and authenticator (Bound Authenticating)

roaming authenticator, although in the conventional method the different protocols are implemented.

4.1 Procedure

4.1.1 Setup

It is necessary that the module of authenticator, AMS and UAF client is installed or updated with digital certificates. For these

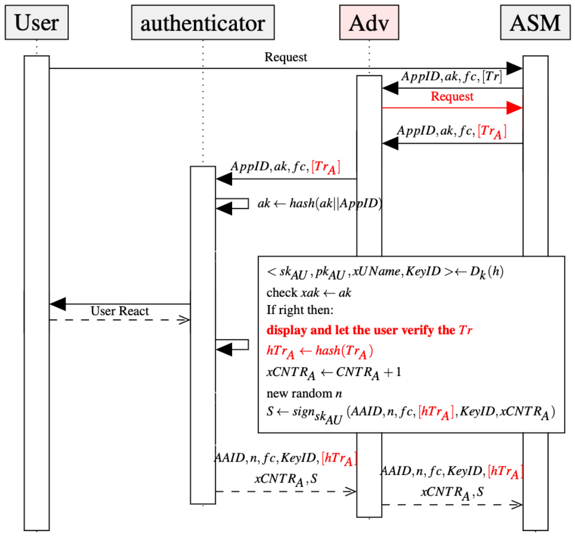
User
authenticator
Adv
ASM
Request
$AppID, ak, fc, [Tr]$
Request
$AppID, ak, fc, [Tr_A]$
$AppID, ak, fc, [Tr_A]$
$ak \leftarrow hash(ak||AppID)$
$< sk_{AU}, pk_{AU}, xUName, KeyID > \leftarrow D_k(h)$
check $xak \leftarrow ak$
If right then:
display and let the user verify the Tr
$hTr_A \leftarrow hash(Tr_A)$
$xCNTR_A \leftarrow CNTR_A + 1$
new random n
$S \leftarrow sign_{sk_{AU}}(AAID, n, fc, [hTr_A], KeyID, xCNTR_A)$
User React
$AAID, n, fc, KeyID, [hTr_A]$
$xCNTR_A, S$
$AAID, n, fc, KeyID, [hTr_A]$
$xCNTR_A, S$

Fig. 3 Malware between ASM and authenticator (Roaming Authenticating)

products created for FIDO authentication itself, the FIDO alliance is basically responsible for issuing the certificates. However, since the authenticator is closely tied to the operating system, there is a possibility that the vendor providing the device manages the certificate. Currently, to provide FIDO authentication as a service, it is necessary to obtain authorization from the FIDO alliance to ensure compliance with the standard protocol. The proposed scheme adds the role of FIDO alliance to issue a certificate for each source file under that authorization process.

In the model specification, in addition to the conventional scheme, let $ASMID$ and $AAID_C$ be a certificate for ASM module and for authenticator module respectively, which are issued by

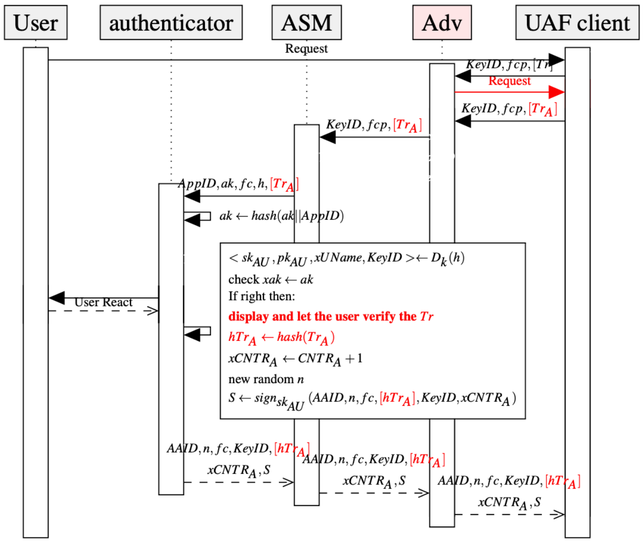
User
authenticator
ASM
Adv
UAF client
Request
$KeyID, fcp, [Tr]$
Request
$KeyID, fcp, [Tr_A]$
$KeyID, fcp, [Tr_A]$
$AppID, ak, fc, h, [Tr_A]$
$ak \leftarrow hash(ak||AppID)$
$< sk_{AU}, pk_{AU}, xUName, KeyID > \leftarrow D_k(h)$
check $xak \leftarrow ak$
If right then:
display and let the user verify the Tr
$hTr_A \leftarrow hash(Tr_A)$
$xCNTR_A \leftarrow CNTR_A + 1$
new random n
$S \leftarrow sign_{sk_{AU}}(AAID, n, fc, [hTr_A], KeyID, xCNTR_A)$
User React
$AAID, n, fc, KeyID, [hTr_A]$
$xCNTR_A, S$
$AAID, n, fc, KeyID, [hTr_A]$
$xCNTR_A, S$
$AAID, n, fc, KeyID, [hTr_A]$
$xCNTR_A, S$

Fig. 4 Malware between ASM and UAF client (Roaming Authenticating)

the FIDO alliance.

4.1.2 Registration Phase

Figure 7 describes the registration phase protocol. The procedure until that the user sends a request and the series of ID information is transmitted to the user agent((*) in Figure 7) is the same for the conventional method. Afterwards, the UAF client sends its own *CallerID* to the predecessor user agent. The user agent then verifies this *CallerID*. Since *CallerID* is a digital certificate, it is checked with the use of a trusted certificate list built into the device beforehand, similar to the procedure for a normal SSL/TLS certificate. If the verification is successful, the user agent and the UAF client follow the conventional process. Then, the the UAF client requests the ASM to send $ASMID$. After the UAF client verifies its validity, the UAF client and the ASM follow the conventional procedure. Note that ak is not generated here, while being used in the conventional method. Next, the ASM verifies the certificate of the authenticator $AAID_C$. If the verification is confirmed, then the same procedure as in the conventional method is conducted until the verification operation of the RP server.

4.1.3 Authentication Phase

Figure 8 describes the authentication phase protocol. The same operation as in the conventional scheme is performed until the UAF client receives the information calculates fcp((*) in Figure 8). Afterwards, the UAF client sends its own *CallerID* to the subsequent ASM. The ASM then verifies this *CallerID*. As in the registration phase, *CallerID* is checked with the trusted certificate list built into the device beforehand. If the verification is successful, the ASM calculates the fc. Then, the fc and the ASM's own certificate $ASMID$, are sent to the authenticator. Next, the authenticator verifies the $ASMID$ received from the ASM using the certificate list. If it is verified, then it proceeds to the next step. In the conventional scheme, there is a process to check the ak sent from the ASM against the ak stored in the registration stage, but this is not done in the proposed scheme because the ASM is authenticated with $ASMID$. After that, there is no change from the conventional process.

4.2 Security Analysis

This subsection explains the mechanism through which the proposed method mitigates the attacks described in section 3. First, for the mis-binding attack at the registration stage, if a malicious software exists before the legitimate UAF client, the attack can be detected when the user agent verifies the *CallerID* from the attacker (malicious UAF client), because the malicious *CallerID* is not attributed to a trusted certificate. Similarly, if the attacker shows up before the legitimate ASM, then the verification of $ASMID$ at the legitimate UAF client detects it. Also for the authenticator, a malware predecessor to the legitimate authenticator is detected when the legitimate ASM verifies $AAID_C$

Regarding an attack in the authentication phase, we first describe the parallel session attack that establishes fraudulent transactions. In the parallel session attack, the values (ak and h) used for authentication of each software module are obtained in advance, and the attack succeeded by re-generating the random numbers used for each session under the conventional scheme. While in the proposed method, the UAF client, ASM, and authenticator are each followed by a subsequent module that verifies its validity based on the certificate confirmation for each session. Specifically, if a malicious file exists before the legitimate ASM, the subsequent ASM can detect it by verifying *CallerID*. If the attacker exists between the legitimate ASM and the legitimate authenticator, then the subsequent legitimate authenticator notices it by verifying $ASMID$.

4.3 Malware Infection Route

The operations described above are performed by malware infection among the legitimate UAF client, the legitimate AMS, and the legitimate authenticator, i.e., the cases when these malicious operation code files are installed between the legitimates. Thus, it is minimum required to check whether the original file is legitimate when it is installed or updated. In other words, in order to reduce the verification time, it may be sufficient to check only when the application file is installed. However, it may have the ability of root exploits. If the malicious file has the root privileges, then the malicious file, which may be installed at any time, enables the manipulation of files involved in FIDO authentication. In this case, it is necessary to authenticate the code file involved in FIDO authentication in each session of FIDO authentication. In fact, the existence of malware capable of root exploits has already been pointed out [15]. When more sensitive information is to be handled using FIDO authentication, the certificate of each code file must be verified in each session.

5. Related Literature

Although FIDO authentication is implemented into various commercial applications, not much academic analysis on it has been conducted. Feng et al. (2021) [5] formulated the UAF protocol and firstly conducted its cryptographic analysis with Proverif. They presented the attacks: mis-binding attack, parallel session attack and DoS attack from the cryptographic aspect and recommended to implement a mechanism of authenticating the UAF client, ASM and Authentiactor as the countermeasure, although they did not provide the specific method. Hu and Zhang (2016) [7] also formulated the UAF protocol and presented two attacks, which are examples of a mis-binding attack and a parallel session attack respectively. However, there is no mention of the countermeasure against the attacks. Panos et al. (2017)[11] listed possible attacks on FIDO authentication: malware infection of each software module, failure of key management and malware infection of OS. They provided the comprehensive security analysis on the scheme, however, none of the specific attacking route or the countermeasure was described. As for attacks in which transaction details are falsified, a vulnerability in the display screen was shown, for which the solution was to place the module that manipulates the screen in the TEE (Trusted Execution Environment) [14], although no methods was included to cover the other attacks such as mis-binding attacks and parallel session attacks.

For FIDO authentication, vulnerabilities other than the FIDO specialised protocol were also shown. Several previous studies show the possibility of attacks on USB keys used for authentication[13] [9]. In addition, re-keying attacks were described, which occurred in authenticating a second factor[4]. Not only regarding FIDO authentication, but also related to the other common authenticating methods in the Internet, the vulnerability on TLS channel, a technical component constructing FIDO authentication was provided in [3].

6. Future Work and Conclusion

This manuscript explains the new attacking paths of the misbinding attack and the parallel session attack. Moreover, we provide the countermeasure against the attacks, in which the certificates on the code file of all software modules are verified in each session or transaction. Since the proposed method is based on a digital certificate attached to the source file, it is not feasible if the certificate itself is not reliable. In fact, there is a possibility of MITM certificate attacks and MITM key attacks. For future direction, methods to overcome these attacks are expected to be provided.

References

[1] Alliance, F. Fido uaf protocol specification, ". *FIDO Alliance Review Draft* (2017).

[2] Baghdasaryan, D., Hill, B., Sasson, R., Hodges, J., and Yang, K. Fido uaf authenticator-specific module api, 2013.

[3] Bhargavan, K., Blanchet, B., and Kobeissi, N. Verified models and reference implementations for the tls 1.3 standard candidate. In *2017 IEEE Symposium on Security and Privacy (SP)* (2017), IEEE, pp. 483–502.

[4] Chang, D., Mishra, S., Sanadhya, S. K., and Singh, A. P. On making u2f protocol leakage-resilient via re-keying. *IACR Cryptol. ePrint Arch. 2017* (2017), 721.

[5] Feng, H., Li, H., Pan, X., and Zhao, Z. A formal analysis of the fido uaf protocol. In *Proceedings of 28th Network And Distributed System Security Symposium (NDSS)* (2021).

[6] Fratantonio, Y., Qian, C., Chung, S. P., and Lee, W. Cloak and dagger: from two permissions to complete control of the ui feedback loop. In *2017 IEEE Symposium on Security and Privacy (SP)* (2017), IEEE, pp. 1041–1057.

[7] Hu, K., and Zhang, Z. Security analysis of an attractive online authentication standard: Fido uaf protocol. *China Communications 13*, 12 (2016), 189–198.

[8] Huang, L.-S., Moshchuk, A., Wang, H. J., Schecter, S., and Jackson, C. Clickjacking: Attacks and defenses. In *21st {USENIX} Security Symposium ({USENIX} Security 12)* (2012), pp. 413–428.

[9] Jacomme, C., and Kremer, S. An extensive formal analysis of multifactor authentication protocols. *ACM Transactions on Privacy and Security (TOPS) 24*, 2 (2021), 1–34.

[10] Niemietz, M., and Schwenk, J. Ui redressing attacks on android devices. *Black Hat Abu Dhabi* (2012).

[11] Panos, C., Malliaros, S., Ntantogian, C., Panou, A., and Xenakis, C. A security evaluation of fido ' s uaf protocol in mobile and embedded devices. In *International Tyrrhenian Workshop on Digital Communication* (2017), Springer, pp. 127–142.

[12] Srinivas, S., Balfanz, D., Tiffany, E., Czeskis, A., and Alliance, F. Universal 2nd factor (u2f) overview. *FIDO Alliance Proposed Standard 15* (2015).

[13] Tian, J., Scaife, N., Kumar, D., Bailey, M., Bates, A., and Butler, K. Sok:"" plug & pray"" today–understanding usb insecurity in versions 1 through c. In *2018 IEEE Symposium on Security and Privacy (SP)* (2018), IEEE, pp. 1032–1047.

[14] Zhang, Y., Wang, X., Zhao, Z., and Li, H. Secure display for fido transaction confirmation. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy* (2018), pp. 155–157.

[15] Zhou, Y., and Jiang, X. Dissecting android malware: Characterization and evolution. In *2012 IEEE symposium on security and privacy* (2012), IEEE, pp. 95–109.

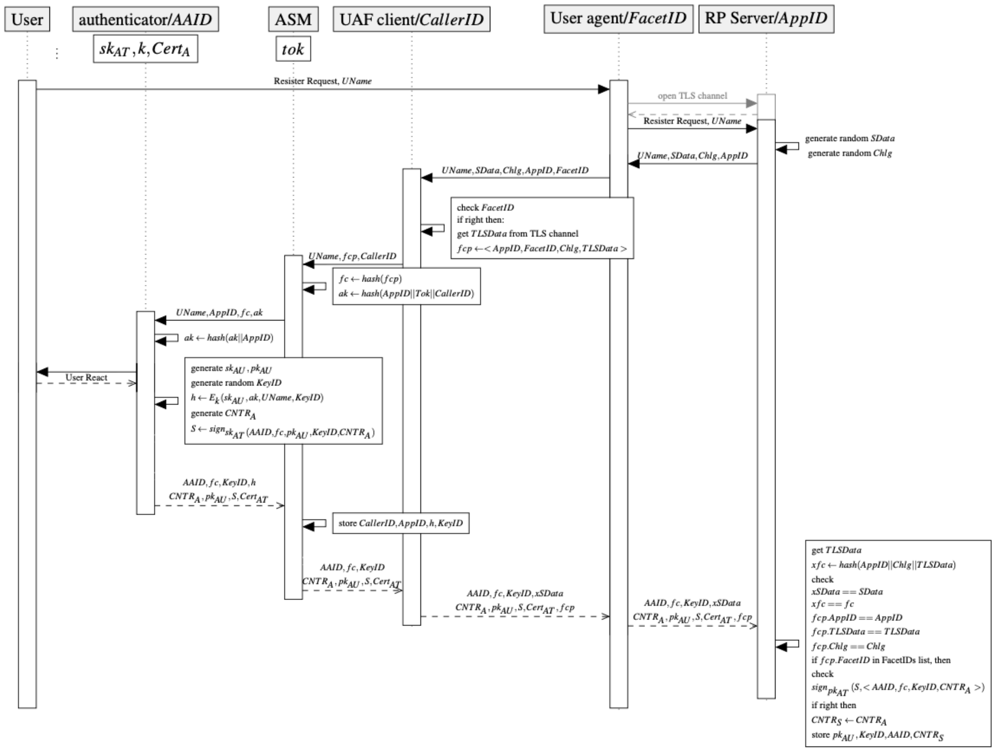
User
authenticator/*AAID*
ASM
UAF client/*CallerID*
User agent/*FacetID*
RP Server/*AppID*
$sk_{AT}, k, Cert_A$
tok
Resister Request, *UName*
open TLS channel
Resister Request, *UName*
generate random *SData*
generate random *Chlg*
UName,SData,Chlg,AppID
UName,SData,Chlg,AppID,FacetID
check *FacetID*
if right then:
get *TLSData* from TLS channel
$fcp \leftarrow < AppID, FacetID, Chlg, TLSData >$
UName, fcp, CallerID
$fc \leftarrow hash(fcp)$
$ak \leftarrow hash(AppID||Tok||CallerID)$
UName,AppID, fc, ak
$ak \leftarrow hash(ak||AppID)$
User React
generate sk_{AU}, pk_{AU}
generate random *KeyID*
$h \leftarrow E_k(sk_{AU}, ak, UName, KeyID)$
generate $CNTR_A$
$S \leftarrow sign_{sk_{AT}}(AAID, fc, pk_{AU}, KeyID, CNTR_A)$
AAID, fc, KeyID, h
$CNTR_A, pk_{AU}, S, Cert_{AT}$
store *CallerID,AppID,h,KeyID*
AAID, fc, KeyID
$CNTR_A, pk_{AU}, S, Cert_{AT}$
AAID, fc, KeyID,xSData
$CNTR_A, pk_{AU}, S, Cert_{AT}, fcp$
AAID, fc, KeyID,xSData
$CNTR_A, pk_{AU}, S, Cert_{AT}, fcp$
get *TLSData*
$xfc \leftarrow hash(AppID||Chlg||TLSData)$
check
$xSData == SData$
$xfc == fc$
$fcp.AppID == AppID$
$fcp.TLSData == TLSData$
$fcp.Chlg == Chlg$
if *fcp.FacetID* in FacetIDs list, then
check
$sign_{pk_{AT}}(S, < AAID, fc, KeyID, CNTR_A >)$
if right then
$CNTR_S \leftarrow CNTR_A$
store $pk_{AU}, KeyID, AAID, CNTR_S$

Fig. 5 Registration Process of the Conventional Scheme

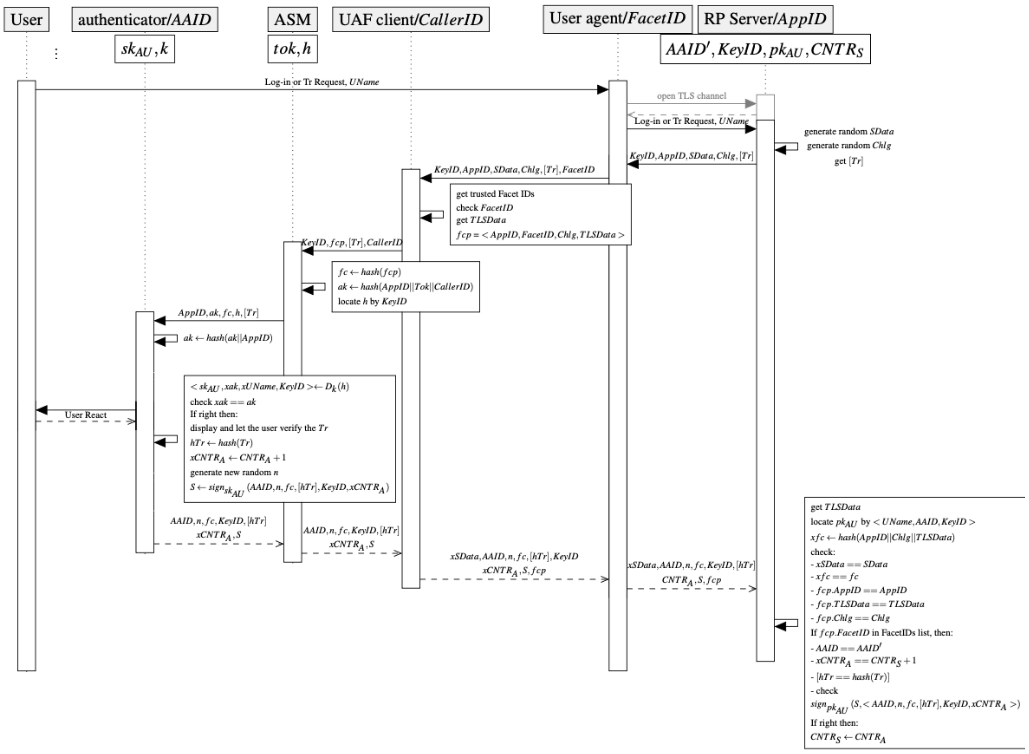
User
authenticator/*AAID*
ASM
UAF client/*CallerID*
User agent/*FacetID*
RP Server/*AppID*
sk_{AU}, k
tok, h
$AAID', KeyID, pk_{AU}, CNTR_S$
Log-in or Tr Request, *UName*
open TLS channel
Log-in or Tr Request, *UName*
generate random *SData*
generate random *Chlg*
get $[Tr]$
$KeyID, AppID, SData, Chlg, [Tr]$
$KeyID, AppID, SData, Chlg, [Tr], FacetID$
get trusted Facet IDs
check *FacetID*
get *TLSData*
$fcp = < AppID, FacetID, Chlg, TLSData >$
$KeyID, fcp, [Tr], CallerID$
$fc \leftarrow hash(fcp)$
$ak \leftarrow hash(AppID||Tok||CallerID)$
locate *h* by *KeyID*
$AppID, ak, fc, h, [Tr]$
$ak \leftarrow hash(ak||AppID)$
User React
$< sk_{AU}, xak, xUName, KeyID > \leftarrow D_k(h)$
check $xak == ak$
If right then:
display and let the user verify the *Tr*
$hTr \leftarrow hash(Tr)$
$xCNTR_A \leftarrow CNTR_A + 1$
generate new random *n*
$S \leftarrow sign_{sk_{AU}}(AAID, n, fc, [hTr], KeyID, xCNTR_A)$
$AAID, n, fc, KeyID, [hTr]$
$xCNTR_A, S$
$AAID, n, fc, KeyID, [hTr]$
$xCNTR_A, S$
$xSData, AAID, n, fc, [hTr], KeyID$
$xCNTR_A, S, fcp$
$xSData, AAID, n, fc, KeyID, [hTr]$
$CNTR_A, S, fcp$
get *TLSData*
locate pk_{AU} by $< UName, AAID, KeyID >$
$xfc \leftarrow hash(AppID||Chlg||TLSData)$
check:
- $xSData == SData$
- $xfc == fc$
- $fcp.AppID == AppID$
- $fcp.TLSData == TLSData$
- $fcp.Chlg == Chlg$
If *fcp.FacetID* in FacetIDs list, then:
- $AAID == AAID'$
- $xCNTR_A == CNTR_S + 1$
- $[hTr == hash(Tr)]$
- check
$sign_{pk_{AU}}(S, < AAID, n, fc, [hTr], KeyID, xCNTR_A >)$
If right then:
$CNTR_S \leftarrow CNTR_A$

Fig. 6 Authentication Process of the Conventional Scheme

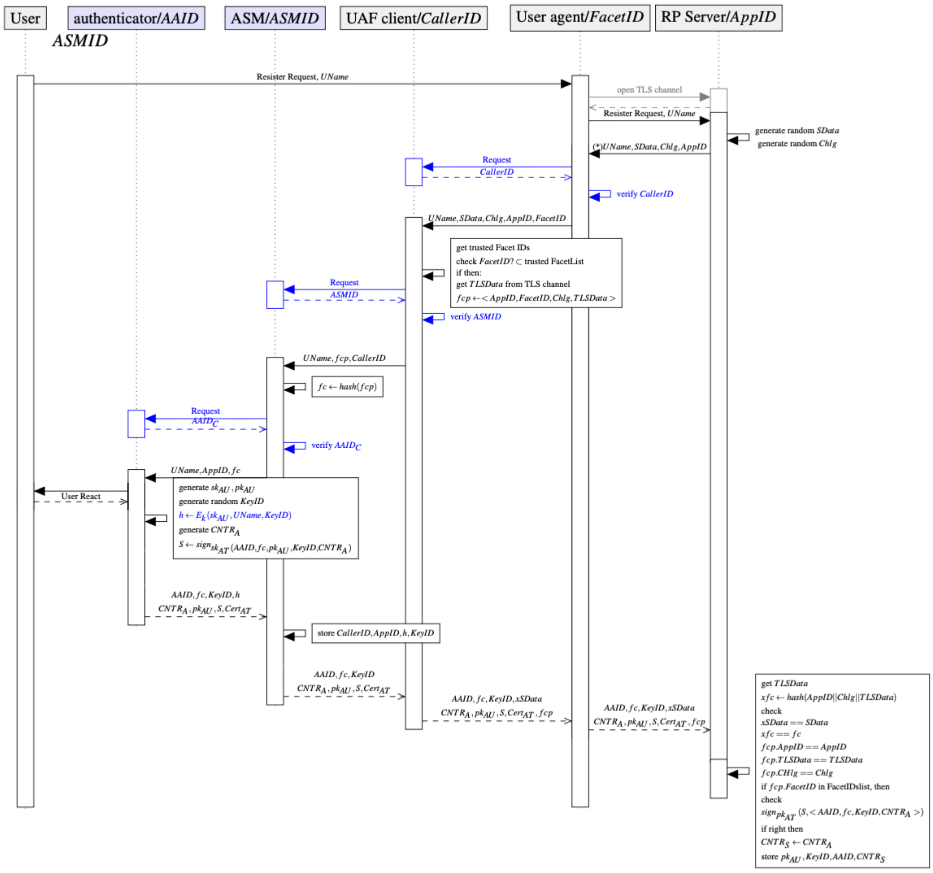
User
authenticator/AAID
ASM/ASMID
UAF client/CallerID
User agent/FacetID
RP Server/AppID
ASMID
Resister Request, UName
open TLS channel
Resister Request, UName
generate random SData
generate random Chlg
(*)UName,SData,Chlg,AppID
Request
CallerID
verify CallerID
UName,SData,Chlg,AppID,FacetID
get trusted Facet IDs
check FacetID? ⊂ trusted FacetList
if then:
get TLSData from TLS channel
$fcp \leftarrow < AppID, FacetID, Chlg, TLSData >$
Request
ASMID
verify ASMID
UName, fcp, CallerID
$fc \leftarrow hash(fcp)$
Request
$AAID_C$
verify $AAID_C$
UName, AppID, fc
User React
generate sk_{AU}, pk_{AU}
generate random KeyID
$h \leftarrow E_k(sk_{AU}, UName, KeyID)$
generate $CNTR_A$
$S \leftarrow sign_{sk_{AT}}(AAID, fc, pk_{AU}, KeyID, CNTR_A)$
AAID, fc, KeyID, h
$CNTR_A, pk_{AU}, S, Cert_{AT}$
store CallerID, AppID, h, KeyID
AAID, fc, KeyID
$CNTR_A, pk_{AU}, S, Cert_{AT}$
AAID, fc, KeyID, xSData
$CNTR_A, pk_{AU}, S, Cert_{AT}, fcp$
AAID, fc, KeyID, xSData
$CNTR_A, pk_{AU}, S, Cert_{AT}, fcp$
get TLSData
$xfc \leftarrow hash(AppID||Chlg||TLSData)$
check
$xSData == SData$
$xfc == fc$
$fcp.AppID == AppID$
$fcp.TLSData == TLSData$
$fcp.CHlg == Chlg$
if fcp.FacetID in FacetIDslist, then
check
$sign_{pk_{AT}}(S, < AAID, fc, KeyID, CNTR_A >)$
if right then
$CNTR_S \leftarrow CNTR_A$
store $pk_{AU}, KeyID, AAID, CNTR_S$

Fig. 7 Registration Process of the Proposed Scheme

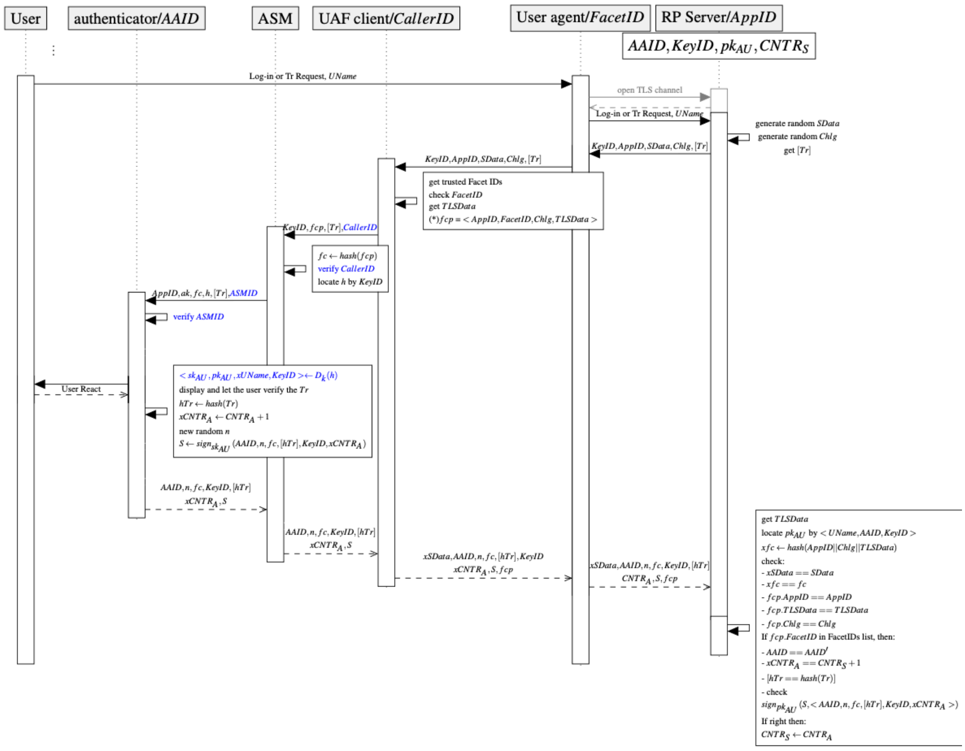
User
authenticator/AAID
ASM
UAF client/CallerID
User agent/FacetID
RP Server/AppID
$AAID, KeyID, pk_{AU}, CNTR_S$
⋮
Log-in or Tr Request, UName
open TLS channel
Log-in or Tr Request, UName
generate random SData
generate random Chlg
get [Tr]
KeyID, AppID, SData, Chlg, [Tr]
KeyID, AppID, SData, Chlg, [Tr]
get trusted Facet IDs
check FacetID
get TLSData
$(*)fcp = < AppID, FacetID, Chlg, TLSData >$
KeyID, fcp, [Tr], CallerID
$fc \leftarrow hash(fcp)$
verify CallerID
locate h by KeyID
AppID, ak, fc, h, [Tr], ASMID
verify ASMID
User React
$< sk_{AU}, pk_{AU}, xUName, KeyID > \leftarrow D_k(h)$
display and let the user verify the Tr
$hTr \leftarrow hash(Tr)$
$xCNTR_A \leftarrow CNTR_A + 1$
new random n
$S \leftarrow sign_{sk_{AU}}(AAID, n, fc, [hTr], KeyID, xCNTR_A)$
AAID, n, fc, KeyID, [hTr]
$xCNTR_A, S$
AAID, n, fc, KeyID, [hTr]
$xCNTR_A, S$
xSData, AAID, n, fc, [hTr], KeyID
$xCNTR_A, S, fcp$
xSData, AAID, n, fc, KeyID, [hTr]
$CNTR_A, S, fcp$
get TLSData
locate pk_{AU} by $< UName, AAID, KeyID >$
$xfc \leftarrow hash(AppID||Chlg||TLSData)$
check:
- $xSData == SData$
- $xfc == fc$
- $fcp.AppID == AppID$
- $fcp.TLSData == TLSData$
- $fcp.Chlg == Chlg$
If fcp.FacetID in FacetIDs list, then:
- $AAID == AAID'$
- $xCNTR_A == CNTR_S + 1$
- $[hTr == hash(Tr)]$
- check
$sign_{pk_{AU}}(S, < AAID, n, fc, [hTr], KeyID, xCNTR_A >)$
If right then:
$CNTR_S \leftarrow CNTR_A$

Fig. 8 Authentication Process of the Proposed Scheme