# A Formal Analysis of the FIDO UAF Protocol

Haonan Feng[†], Hui Li[†‡], Xuesong Pan[†] and Ziming Zhao[⋆]
[†]Beijing University of Posts and Telecommunications [⋆]CactiLab, University at Buffalo
[†]{fenghaonan, lihuill, panxuesong}@bupt.edu.cn [⋆]zimingzh@buffalo.edu

***Abstract*—The FIDO protocol suite aims at allowing users to log in to remote services with a local and trusted authenticator. With FIDO, relying services do not need to store user-chosen secrets or their hashes, which eliminates a major attack surface for e-business. Given its increasing popularity, it is imperative to formally analyze whether the security promises of FIDO hold. In this paper, we present a comprehensive and formal verification of the FIDO UAF protocol by formalizing its security assumptions and goals and modeling the protocol under different scenarios in ProVerif. Our analysis identifies the minimal security assumptions required for each of the security goals of FIDO UAF to hold. We confirm previously manually discovered vulnerabilities in an automated way and disclose several new attacks. Guided by the formal verification results we also discovered 2 practical attacks on 2 popular Android FIDO apps, which we responsibly disclosed to the vendors. In addition, we offer several concrete recommendations to fix the identified problems and weaknesses in the protocol.**

## I. Introduction

Fast IDentity Online (FIDO) has gained significant popularity in recent years as a public-key cryptography based authentication framework that enables users to login *remote* online services and websites by authenticating themselves to *local trusted* authenticators, such as a fingerprint scanner on a smartphone. With FIDO, relying web services do not need to store user-chosen secrets or their hashes, which eliminates a major attack surface for e-business [26], [27], [55]. At the time of writing, more than 250 companies have become members of the FIDO alliance [5], and more than 703 certified FIDO products are in the market [15]. Android 7.0+ is now also FIDO2 certified out of the box, and Microsoft Windows has been supporting the FIDO2 standard since October 2018, which gives billions of users the ability to leverage built-in authenticators for passwordless access to websites and applications [2].

The original FIDO protocol suite consists of two sets of specifications: Universal Authentication Framework (UAF) and Universal Second Factor (U2F). UAF allows users to register their accounts with the relying party through a trusted authenticator and replaces the traditional password login scheme. U2F allows users to add a second-factor local authenticator to enhance the security of their accounts. FIDO2 was officially launched in 2018 with the addition of Web Authentication specification (WebAuthn) [52] and Client-to-Authenticator Protocol (CTAP) [3]. WebAuthn supports online services to use FIDO through a standardized web API, whereas CTAP supports external devices to work with browsers supporting WebAuthn.

Given the increasing popularity of FIDO, it is imperative to analyze if its security promises hold. Some flaws of FIDO have already been identified using manual analysis [29], [36], [42], [43], [47]. Even though these ad hoc methods can help discover some vulnerabilities, they lack a formal foundation and are not capable of systematically verifying the properties of FIDO.

Also, there have been several attempts to formally verify FIDO [38], [48]. However, they have many limitations: i) none of them presented a formalization of the security assumptions and goals from the FIDO specifications, which led to an inaccurate, if not incorrect, modeling of the protocol and security properties; ii) they focused on the U2F protocol, which has a simpler attack model than the UAF protocol does. This is because multiple vulnerable entities in the UAF are consolidated into one trusted physical device in the U2F; iii) their oversimplified modeling of the protocol failed in discovering more vulnerabilities.

In fact, formally verifying the security properties of the UAF protocol is challenging: i) many security assumptions and security goals are implicit and buried in over 500 pages of English specifications across 19 documents. The formal extraction of them requires considerable analysis and interpretation of the specifications; ii) the attack model is complicated because many entities in the protocol can be compromised in real-world settings. A comprehensive verification should consider all possible scenarios; iii) the UAF protocol is complex with many steps and optional steps, which should also be considered in verification.

To systematically evaluate the security of UAF, we tackle aforementioned challenges and resort to formal methods, which have been used in verifying the security of real-world protocols, such as Needham-Schroeder [44], TLS [20]–[22], 5G authentication [18], IKE [30], Diffie-Hellman [1], [49], ISO/IEC 9798-2 [57], LMAP [37], vTPM migration [28], 3PAKE [54], e-voting [32], E-Health [33], USB Type-C [50], etc. The contributions of this paper are summarized as follows:

1) We provide a *formalization of UAF's security assumptions and goals* by extracting and formally interpreting them from the specifications. We consider all sorts of properties, including authentication, non-repudiation, confidentiality, and privacy;

2) We provide a *faithful and formal model of the UAF protocol*, which is a pragmatic balance of a detailed representation

[‡]Corresponding author

Network and Distributed Systems Security (NDSS) Symposium 2021
21-25 February 2021, Virtual
ISBN 1-891562-66-5
https://dx.doi.org/10.14722/ndss.2021.24363
www.ndss-symposium.org

that can lead to the discovery of many vulnerabilities and an abstraction that is amenable to formal verification tools;

3) We carry out an *automatic verification in the symbolic model* using ProVerif [23], [24]. We open-source our verification tool and code UAFVerif[1];

4) We present the *verification results* by confirming previously found vulnerabilities and disclosing new attacks. Also, we present the *minimal assumptions* for UAF to meet each security property;

5) Guided by the analysis results, we perform 2 attacks on 2 popular Android UAF apps, which represent two types of vulnerabilities. We responsibly disclosed them to the vendors, resulting in 1 vulnerability ID (CNNVD-202005-1219);

6) We offer *several concrete and explicit suggestions* to fix the identified problems in the protocol and specifications.

The rest of the paper is organized as follows. Section II introduces the UAF architecture and protocol. Section III presents our interpretation and formal model of UAF's security assumptions and goals. In Section IV, we explain the basic principle of ProVerif and our modeling choices. We present the security analysis results of the UAF protocol using ProVerif in Section V and summarize the defects of the UAF protocol. Besides, we discuss some possible attacks for the UAF protocol and give some recommendations. We introduce the related works in Section VI. Section VII concludes the paper.

## II. Overview of the UAF Protocol

The UAF protocol has two major operations, namely authenticator registration and authentication. At a high level, the UAF protocol works as follows: a user wishes to log in to remote services using a device that has a certified UAF authenticator, e.g., fingerprint sensor. The authenticator has a trusted *attestation key* (either RSA or ECDSA). The user logs in to a relying party, such as a banking website, using her original credentials, e.g., text-based password. The authenticator records her fingerprint, generates an *authentication key* for this website, signs the public part of the new key with the attestation key, and sends it to the website. The website links the user's online profile with the authentication key if it is valid. As a result, the trust between the relying party and the authenticator is established and the procedure of *authenticator registration* is completed. In subsequent login attempts (the *authentication* procedure), the user only needs to prove her identity to the local authenticator, upon the success of which the website and the authenticator will run a challenge-response protocol with the authentication key.

Table I describes the acronyms used in this paper. Section II-A presents the overall architecture and entities of UAF. Some of the steps and exchanged messages of the protocol differ based on the type of authenticator in use. In Section II-B and II-C, we illustrate the protocol using 1st-factor bound authenticators [7]. Section II-D explains the protocol operations under different types of authenticators and use cases.

### A. Architecture

As shown in Figure 1, we abstract 5 major entities and 4 communication channels in the Universal Authentication

[1]https://github.com/CactiLab/UAFVerif

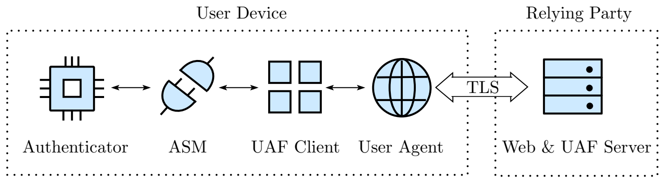

Fig. 1. UAF architecture

Framework:

1) an *authenticator*, which has an internal matcher for user verifications, stores a model identifier, an attestation key, and a symmetric key $\langle AAID, sk_{AT}, k_W \rangle$. The authenticator generates asymmetric authentication keys ($sk_{AU}$, $pk_{AU}$) [8]. There are 4 types of authenticators, 1st-factor bound authenticator (1B), 2nd-factor bound authenticator (2B), 1st-factor roaming authenticator (1R), and 2nd-factor roaming authenticator (2R). They differ in generating and using $h$, *KeyID* and $ak$, which we explain in Section II-D;

2) the *Authenticator Specific Module* (ASM) is an authenticator abstraction layer that provides a uniform API for the upper layer. When ASM is launched for the first time, it generates a secret ($tok$) [9];

3) a *UAF Client* (UC) is a system service or application that implements the client-side logic of the UAF protocol. A UAF Client is identified by a *CallerID*, which ASM can retrieve from the operating system. For example, on Android, *CallerID* is the hash of the UAF Client's APK signing certificate;

4) a *User Agent* (UA) is a user application identified by a URI named *FacetID*. When the user agent is a browser, *FacetID* is the web origin of the web page triggering the UAF operation, e.g., `https://login.example.com/`. When the user agent is an app on Android, *FacetID* is the hash of the user agent's APK signing certificate;

5) a *Relying Party* (RP) consists of a web server and a UAF server. The UAF server ensures that only trusted authenticators can be registered, manages the association of authenticators to user accounts, and evaluates user authentications.

The UAF specifications require TLS communication between a UA and an RP. Other software entities communicate via inter-process communication (IPC) methods or hardware and software communication.

### B. Authenticator Registration

Figure 2 depicts the message flows of the UAF authenticator registration operation using a 1B authenticator.

Upon the success of the original authentication method, e.g., text-based password, RP generates a registration request message $\langle$*UName*, *AppID*, *SData*, *Chlg*$\rangle$ and sends it to UC. *UName* identifies the user, while *AppID* is a URL that points to a list of trusted user agents. *Chlg* is a random challenge value, and *SData* is a session identifier created by the Relying Party.

After receiving the request message from RP, UC retrieves the trusted user agent list from *AppID* and verifies if *FacetID* is on the list [4]. UC stores the session ID *SData* as *xSData* and computes *TSLData* using the TLS channel information to

| Acronym | Full name | Description |
|---|---|---|
| *RP* | Relying Party | The server-side, which contains a web server and a UAF server. |
| *UA* | User Agent | A user application that supports the UAF protocol. |
| *UC* | UAF Client | A system service or application that implements the client-side logic of the UAF protocol. |
| *ASM* | Authenticator Specific Module | An authenticator abstraction layer that provides a uniform API for the upper layer. |
| *UName* | Username | A human-readable string identifying a user's account at a Relying Party. |
| *AppID* | Application Identifier | A URL pointing to the trusted facets. |
| *FacetID* | Application Facet Identifier | A platform-specific identifier (URI) for an application facet to indicate how an application is implemented on various platforms (such as Web applications, Android applications or IOS applications). |
| *SData* | Server Data | A session identifier created by the Relying Party. |
| *Chlg* | Server Challenge | A random value that is provided by the FIDO UAF Server in the UAF protocol requests. |
| *Tr* | Transaction Text | Text to be confirmed in the case of transaction confirmation. |
| *TLSData* | Channel Binding | A channel binding allows applications to establish that the two end-points of a secure channel at one network layer are the same as at a higher layer by binding authentication to the higher layer to the channel at the lower layer. |
| *AAID* | Authenticator Attestation Identifier | A unique identifier assigned to a model, class or batch of FIDO UAF Authenticators that all share the same characteristics |
| *CNTR* | Signature counter | A monotonically increasing counter maintained by the Authenticator. It is increased on every use of the Authentication private key. FIDO UAF Server uses this value to detect cloned authenticators. |
| $tok$ | ASMToken | A randomly generated secret when the ASM is launched the first time and the ASM will maintain this secret until the ASM is uninstalled. |
| $ak$ | Key handle access token | An access control mechanism for protecting an authenticator's FIDO UAF credentials from unauthorized use. It is created by the ASM by mixing various sources of information. |
| $fc$ | Final Challenge | The final challenge for the Challenge-Response mechanism. |
| $h$ | KeyHandle | A key container created by a FIDO UAF Authenticator, containing an authentication private key and (optionally) other data (such as Username). |
| *KeyID* | Key Identifier | An opaque identifier for an authentication key registered by an authenticator with a FIDO UAF Server. |
| $sk_{AT}$ | Attestation Private Key | The private asymmetric key used for FIDO UAF Authenticator attestation. |
| $pk_{AT}$ | Attestation Public Key | The public asymmetric key used for FIDO UAF Authenticator attestation. |
| $sk_{AU}$ | Authentication Private Key | User authentication private key generated by FIDO UAF Authenticator. |
| $pk_{AU}$ | Authentication Public Key | User authentication public key generated by FIDO UAF Authenticator. |
| $k_W$ | Wrapping Key | A symmetric key to wrap the data inside the authenticator |

TABLE I. Acronyms and descriptions.

prevent the TLS MIMT attack [39]. Then, UC sends *UName* and the final challenge parameters $fcp = \langle$*AppID*, *FacetID*, *Chlg*, *TLSData*$\rangle$ to ASM.

ASM computes the final challenge $fc = \texttt{hash}\,(fcp)$ and a token $ak = \texttt{hash}\,($*AppID* $||$ $tok$ $||$ *CallerID*$)$, where $||$ denotes concatenation. $ak$ is a token under the *KHAccessToken* mechanism, which is an access control mechanism for protecting an authenticator's FIDO UAF credentials from unauthorized use [9]. The authenticator uses $ak$ in the procedure of authentication to verify ASM. Then, ASM sends $\langle$*UName*, *AppID*, $ak$, $fc\rangle$ to the authenticator.

The authenticator updates the token $ak = \texttt{hash}(ak$ $||$ *AppID*$)$. Then, the authenticator triggers its built-in matcher, e.g., fingerprint sensor, to locally verifies the user's identity. Then, an authentication key pair $\langle sk_{AU}, pk_{AU}\rangle$ for this user account is generated. The authenticator generates a random *KeyID* as the key identifier. The authenticator computes a key handle $h = \texttt{E}_{k_W}(sk_{AU}$, $ak$, *UName*, *KeyID*$)$, where $\texttt{E}_{k_W}$ is the symmetric encryption. After that, the authenticator generates a random signature counter $CNTR_A$, which should be synchronized with RP. *CNTR* can be used by RP to detect cloned authenticators. Finally, the authenticator computes the signature $S = \texttt{sign}_{sk_{AT}}$ (*AAID*, $fc$, *KeyID*, $CNTR_A$, $pk_{AU}$), where $\texttt{sign}$ is a sign function, and sends $\langle$*AAID*, $fc$, *KeyID*, $CNTR_A$, $pk_{AU}$, $S\rangle$ to ASM, ASM stores *CallerID*, *AppID*, $h$, *KeyID* and sends the messages to UC. UC forwards the message, *xSData* and *fcp* to RP.

To verify RP is in the same session with UA and UC, RP compares $xfc = \texttt{hash}\,($*AppID* $||$ *Chlg* $||$ *TLSData*$)$ with the received $fc$ and compares *SData* with *xSData*. Next, it verifies *fcp.AppID*, *fcp.Chlg* and *fcp.TLSData* correspond to those stored in RP, and checks if *fcp.FacetID* is in the trusted FacetIDs list. Then, RP verifies the signature $S$ with the attestation public key ($pk_{AT}$) of this authenticator. If the signature matches, RP stores $CNTR_A$, $pk_{AU}$, *KeyID*, *AAID* and completes this registration.

## C. Authentication

Figure 3 depicts the message flows of the authentication operation using a 1B authenticator to step-up authentication, which can also be extended to the transaction confirmation operation. The transaction confirmation offers support for prompting a user to confirm a specific transaction with a secure display device. In Figure 3, the transaction confirmation related operations are marked with a '[]'.

In authentication and transaction confirmation, RP sends an authentication request message $\langle$*AppID*, *KeyID*, *SData*, *Chlg*, [*Tr*]$\rangle$ to UC. Then UC computes *fcp* $= \langle$*AppID*, *Facet*, *Chlg*, *TLSData*$\rangle$, and sends $\langle$*fcp*, *KeyID*, [*Tr*]$\rangle$ to ASM.

Once ASM receives the message, it computes the final challenge $fc = \texttt{hash}\,($*fcp*$)$ and the token $ak = \texttt{hash}\,($*AppID* $||$ $tok$ $||$ *CallerID*$)$, after which it locates $h$ by *KeyID* and sends $\langle ak$, $fc$, *AppID*, $h$, [*Tr*]$\rangle$ to the authenticator.

Upon receiving the message, the authenticator updates the token $ak = \texttt{hash}(ak$ $||$ *AppID*$)$ and triggers its built-in matcher to verifies the user's identity. Then, the authenticator computes $\langle sk_{AU}$, $xak$, *xUName*, *KeyID*$\rangle$ = $\texttt{D}_{k_W}(h)$, where $\texttt{D}_{k_W}$ is the decryption function. Next, the authenticator checks if $xak$ matches $ak$ to make sure it is the trusted ASM. If the check passes, the authenticator displays the transaction text *Tr* on the secure display for the user to confirm. Then, it computes *hTr* = $\texttt{hash}$(*Tr*) and increases $CNTR_A$ to $xCNTR_A$. A random value $n$ is generated to protect the authenticator from the replay attack. Finally, the authenticator computes the signature $S = \texttt{sign}_{sk_{AU}}$ (*AAID*, $n$, $fc$, [*hTr*],

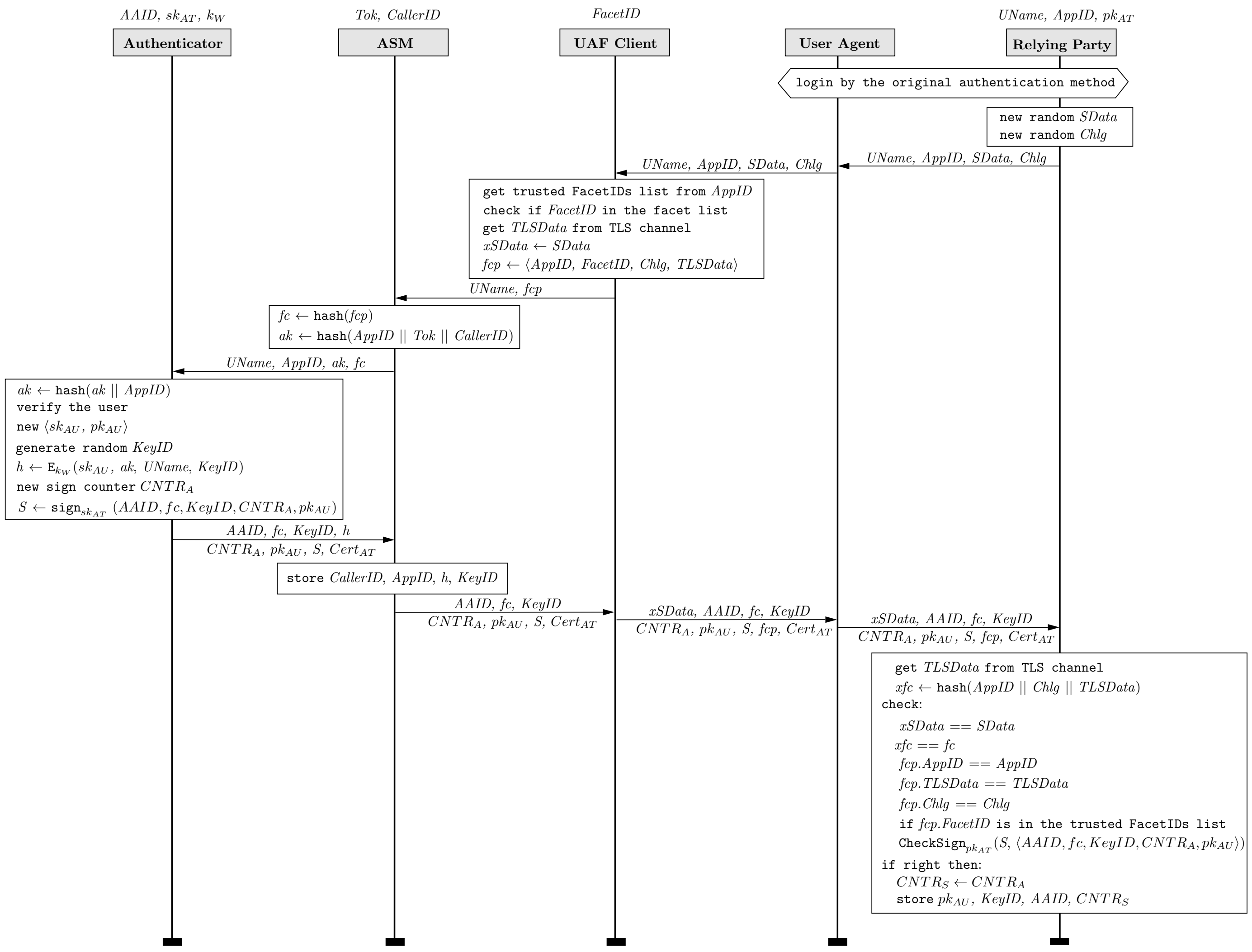
AAID, $sk_{AT}$, $k_W$
Authenticator
Tok, CallerID
ASM
FacetID
UAF Client
User Agent
UName, AppID, $pk_{AT}$
Relying Party
login by the original authentication method
new random SData
new random Chlg
UName, AppID, SData, Chlg
UName, AppID, SData, Chlg
get trusted FacetIDs list from AppID
check if FacetID in the facet list
get TLSData from TLS channel
xSData ← SData
fcp ← ⟨AppID, FacetID, Chlg, TLSData⟩
UName, fcp
fc ← hash(fcp)
ak ← hash(AppID || Tok || CallerID)
UName, AppID, ak, fc
ak ← hash(ak || AppID)
verify the user
new ⟨$sk_{AU}$, $pk_{AU}$⟩
generate random KeyID
h ← $E_{k_W}$($sk_{AU}$, ak, UName, KeyID)
new sign counter $CNTR_A$
S ← $sign_{sk_{AT}}$(AAID, fc, KeyID, $CNTR_A$, $pk_{AU}$)
AAID, fc, KeyID, h
$CNTR_A$, $pk_{AU}$, S, $Cert_{AT}$
store CallerID, AppID, h, KeyID
AAID, fc, KeyID
$CNTR_A$, $pk_{AU}$, S, $Cert_{AT}$
xSData, AAID, fc, KeyID
$CNTR_A$, $pk_{AU}$, S, fcp, $Cert_{AT}$
xSData, AAID, fc, KeyID
$CNTR_A$, $pk_{AU}$, S, fcp, $Cert_{AT}$
get TLSData from TLS channel
xfc ← hash(AppID || Chlg || TLSData)
check:
xSData == SData
xfc == fc
fcp.AppID == AppID
fcp.TLSData == TLSData
fcp.Chlg == Chlg
if fcp.FacetID is in the trusted FacetIDs list
$CheckSign_{pk_{AT}}$(S, ⟨AAID, fc, KeyID, $CNTR_A$, $pk_{AU}$⟩)
if right then:
$CNTR_S$ ← $CNTR_A$
store $pk_{AU}$, KeyID, AAID, $CNTR_S$

Fig. 2. Registration of the Authenticator

*KeyID*, $xCNTR_A$) and sends ⟨*AAID*, $n$, $fc$, [*hTr*], *KeyID*, $xCNTR_A$, $S$⟩ to UC, which sends the message, *xSData* and *fcp* to RP.

RP locates $pk_{AU}$ of this user by ⟨*UName*, *AAID'*, *KeyID*⟩. It compares *SData* and *xSData* to make sure it is the same session. Next, it it verifies *fcp.AppID*, *fcp.Chlg* and *fcp.TLSData* correspond to those stored in RP, and checks if *fcp.FacetID* is in the trusted FacetIDs list. Then, it compares *AAID'* with *AAID* to ensure that the message comes from the same authenticator registered with RP. Then RP computes $xfc$ = hash (*AppID* || *Chlg* || *TLSData*) and compares it with $fc$ to make sure the response is right. Then it compares *hTr* with hash(*Tr*) and verifies the signature $S$. Finally, RP checks whether $xCNTR_A$ increases compared to $CNTR_S$, if not, the sync failed. If all the checks pass, RP updates $CNTR_S$ with $xCNTR_A$ and finishes the authentication process.

## *D. Protocol Operations Under Different Authenticators and Use Cases*

In FIDO, bound authenticators (1B and 2B) are embedded into a user's device, e.g., a built-in fingerprint sensor. Roaming authenticators (1R and 2R) are not bound to any device, e.g., a USB dongle with a built-in capacitive touch device. Users can use roaming authenticators with any number of devices. The 1st-factor authenticators (1B and 1R) normally operate as the first factor to authenticate users, while the 2nd-factor authenticator can operate in multi-factor authentication.

Also, there are two use cases when it comes to an authenticator executing the sign command. One case is that there is no user session (no cookies, a clear machine). In this case, RP communicates with UA for the first time and does not know who the user is and cannot provide *KeyID* associated with the user. In this paper, we call this case login authentication. The other case is called step-up authentication, where there is already a user session in which the user is unauthenticated or nominally authenticated. RP knows who the user is and provides *KeyID* associated with the user. For example, transaction confirmation can only happen when there is a user session.

The UAF protocol under different types of authenticators and use cases differ in some steps and messages [8]. Table II summarizes the differences: i) only 1st-factor authenticators (1B and 1R) can be used in login authentication cases; ii) 2R

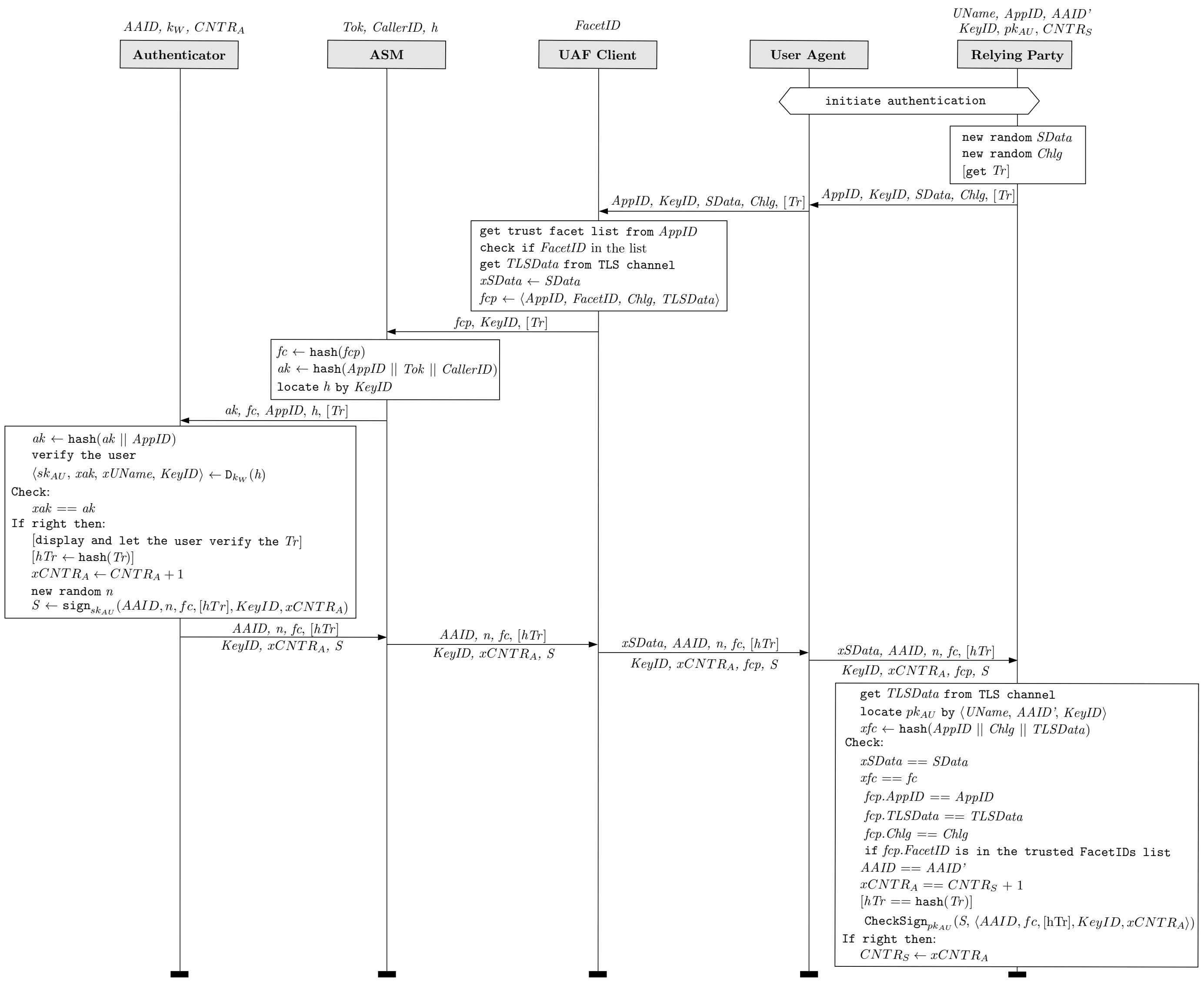
AAID, $k_W$, $CNTR_A$
Authenticator
Tok, CallerID, h
ASM
FacetID
UAF Client
User Agent
UName, AppID, AAID'
KeyID, $pk_{AU}$, $CNTR_S$
Relying Party
initiate authentication
new random SData
new random Chlg
[get Tr]
AppID, KeyID, SData, Chlg, [Tr]
AppID, KeyID, SData, Chlg, [Tr]
get trust facet list from AppID
check if FacetID in the list
get TLSData from TLS channel
$xSData \leftarrow SData$
$fcp \leftarrow \langle AppID, FacetID, Chlg, TLSData \rangle$
fcp, KeyID, [Tr]
$fc \leftarrow$ hash(fcp)
$ak \leftarrow$ hash(AppID || Tok || CallerID)
locate h by KeyID
ak, fc, AppID, h, [Tr]
$ak \leftarrow$ hash(ak || AppID)
verify the user
$\langle sk_{AU}, xak, xUName, KeyID \rangle \leftarrow D_{k_W}(h)$
Check:
$xak == ak$
If right then:
[display and let the user verify the Tr]
[$hTr \leftarrow$ hash(Tr)]
$xCNTR_A \leftarrow CNTR_A + 1$
new random n
$S \leftarrow \mathrm{sign}_{sk_{AU}}(AAID, n, fc, [hTr], KeyID, xCNTR_A)$
AAID, n, fc, [hTr]
KeyID, $xCNTR_A$, S
AAID, n, fc, [hTr]
KeyID, $xCNTR_A$, S
xSData, AAID, n, fc, [hTr]
KeyID, $xCNTR_A$, fcp, S
xSData, AAID, n, fc, [hTr]
KeyID, $xCNTR_A$, fcp, S
get TLSData from TLS channel
locate $pk_{AU}$ by $\langle$UName, AAID', KeyID$\rangle$
$xfc \leftarrow$ hash(AppID || Chlg || TLSData)
Check:
$xSData == SData$
$xfc == fc$
$fcp.AppID == AppID$
$fcp.TLSData == TLSData$
$fcp.Chlg == Chlg$
if fcp.FacetID is in the trusted FacetIDs list
$AAID == AAID'$
$xCNTR_A == CNTR_S + 1$
[$hTr ==$ hash(Tr)]
$\mathrm{CheckSign}_{pk_{AU}}(S, \langle AAID, fc, [hTr], KeyID, xCNTR_A \rangle)$
If right then:
$CNTR_S \leftarrow xCNTR_A$

Fig. 3. Authentication operation: the operations framed by '[]' are needed only in the transaction confirmation operations.

| Authenticator | Authenticator Registration | Authentication |
|---|---|---|
| 1B | random *KeyID*<br>$h = \mathrm{E}_{k_W}(sk_{AU}, ak, UName, KeyID) \rightarrow$ ASM<br>$ak = \mathrm{hash}(AppID \mid\mid tok \mid\mid CallerID)$ | login: RP does not provide *KeyID*<br>step-up: RP provides *KeyID* |
| 2B | random *KeyID*<br>$h = \mathrm{E}_{k_W}(sk_{AU}, ak, KeyID) \rightarrow$ ASM<br>$ak = \mathrm{hash}(AppID \mid\mid tok \mid\mid CallerID)$ | step-up: RP provides *KeyID* |
| 1R | random *KeyID*<br>$h = \mathrm{E}_{k_W}(sk_{AU}, ak, UName, KeyID) \rightarrow$ Authenticator<br>$ak = \mathrm{hash}(AppID)$ | login: RP does not provide *KeyID*<br>step-up: RP provides *KeyID* |
| 2R | *KeyID* = *h*<br>$h = \mathrm{E}_{k_W}(sk_{AU}, ak) \rightarrow$ RP<br>$ak = \mathrm{hash}(AppID)$ | step-up: RP provides *KeyID* |

TABLE II. PROTOCOL DIFFERENCES UNDER DIFFERENT TYPES OF AUTHENTICATORS AND USE CASES.

authenticators use $h$ as *KeyID*, whereas other authenticators generate a random *KeyID*; iii) since bound authenticators do not have internal storage, they store $h$ in ASM. 1R authenticators have internal storage and store $h$ inside themselves. The protocol requires 2R authenticators to store $h$ at RP; iv) if *KeyID* is generated randomly, it is stored in $h$; v) if it is a 1st-factor authenticator (1B and 1R), $h$ contains *UName*; vi) for bound authenticators, ASM generates $ak$ with *AppID*, $tok$ and *CallerID*. For roaming authenticators, $ak$ only contains *AppID*; vii) in the authentication process, RP provides *KeyID* only for step-up authentication cases.

## III. Threat Model and Security Goals

The UAF specifications list their security assumptions in Section 6 of the security reference [6], and provide the allowed cryptography list [11], the allowed operating environment list [12], the authenticator metadata requirements [13], the authenticator security requirements [14] in respective documents. However, the security assumptions are very strong and impractical, and many real-world deployments do not strictly follow them. To provide a more comprehensive analysis, we strive to analyze under the security assumptions that cover more realistic scenarios.

Because automation in the extraction and formalization of security assumptions and goals from lengthy and ambiguous natural language documents is still very challenging, we manually extract the security goals of the UAF from several documents and translate the informal descriptions of the security goals into precise and formal expressions, which would be the precondition for the formal analysis. We use our experience to make choices in what to model and how to model to achieve a balance between analysis feasibility and model accuracy [6]–[9].

### A. Assumptions and Threat Model

*1) Assumptions on Cryptographic Algorithms:* We assume the cryptographic algorithms are secure, which means without knowing the correct keys, the adversary can never forge signatures or decrypt messages.

*2) Assumptions on Channels and Entities:* UAF uses four channels, as shown in Figure 1. The communications among Authenticator, ASM, UC, and UA use interprocess communication (IPC) channels or hardware and software communication channels. The communications between UA and RP use a network channel, which is protected by the TLS. We assume that i) the attacker cannot eavesdrop, intercept, or manipulate the communications on an established channel between legitimate entities; but ii) an attacker can install malicious entities to initiate and accept communication requests. For example, a user may be tricked into installing a malicious UC downloaded from the internet, which could communicate with legitimate UA or ASM. We verify if the security properties of UAF hold when there are different malicious entities.

In addition, applications are subject to known software attacks and may be controlled by attackers (e.g., hook, root), so we verify if the security properties of UAF hold when each of the entities is compromised. We also consider malicious or compromised authenticators because i) in some real-world deployments, the authenticator is implemented as software; ii) hardware-based authenticators are subject to side-channel attacks [40], [41].

*3) Assumptions on Data Protections:* We assume the following data fields are public, and the attacker has access to all of them: *FacetID*, *CallerID*, *AAID*, *AppID*, *UName*, $pk_{AT}$, $pk_{AU}$. We verify if the security properties of UAF hold when the following data is compromised or leaked: $k_W$, $tok$, *CNTR*, $h$, $sk_{AT}$, $sk_{AU}$.

*4) Assumptions on Authenticators:* We assume the authenticator always authenticates users correctly, RP authenticating the authenticator is equivalent to RP authenticating the user. Because a large number of authenticators may share the same *AAID* and $sk_{AT}$ for privacy preserving [10], we assume the attacker has an authenticator with the same $sk_{AT}$ and *AAID* as the user's, with which the attacker can calculate the signature and pass the authentication of RP.

### B. Formalization of UAF's Security Goals

The formal expressions are indicated in purple text.

*1) Authentication Properties:* To precisely formalize the authentication properties, we resort to Lowe's taxonomy of authentication properties [45], which can be directly modeled in formal methods and widely used in previous research [18]. Lowe's taxonomy specifies multiple levels of authentication, from A's point of view, between an initiator A and a responder B: i) aliveness: whenever A completes a run of the protocol, the aliveness property ensures that B has previously been running the protocol, but not necessarily with A; ii) weak agreement: whenever A completes a run of the protocol, the weak agreement property ensures that B has previously been running the protocol with A, but not necessarily with the same data; iii) non-injective agreement on data items $ds$: whenever A completes a run of the protocol, the non-injective agreement property ensures that B has previously been running the protocol with A. Besides, A and B agreed on the data values in $ds$. However, the property cannot guarantee that there is a one-to-one relationship between the runs of A and the runs of B; iv) injective agreement on data items $ds$: whenever A completes a run of the protocol, the injective agreement property ensures that B has previously been running the protocol with A. Besides, A and B agreed on the data values in $ds$, and each such run of A corresponds to a unique run of B. This prevents replay attacks.

The authentication goals are extracted from Section 4 of the security reference [6]. The overall goal of the UAF is SG-1.

**SG-1 Strong User Authentication:** Authenticate (i.e. recognize) a user and/or a device to a Relying Party with high (cryptographic) strength.

After a successful authentication process, RP (identified by *AppID*) shall authenticate a user account (identified by *UName*) with a unique registered authentication key pair (identified by *KeyID*) generated by a registered authenticator (identified by *AAID*). Formally, RP must obtain non-injective agreement on *UName, AAID, KeyID, AppID* with the authenticator after the authentication process.

The authentication goals are also complemented by SG-10, SG-11, SG-12, and SG-13.

**SG-10 DoS Resistance:** Be resilient to Denial of Service Attacks. I.e. prevent attackers from inserting invalid registration information for a legitimate user for the next login phase. Afterward, the legitimate user will not be able to login successfully anymore.

**SG-11 Forgery Resistance**: Be resilient to Forgery Attacks (Impersonation Attacks). I.e. prevent attackers from attempting to modify intercepted communications to masquerade as the legitimate user and log into the system.

**SG-12 Parallel Session Resistance**: Be resilient to parallel Session Attacks. Without knowing a user's authentication credential, an attacker can masquerade as the legitimate user by creating a valid authentication message out of some eavesdropped communication between the user and the server.

**SG-13 Forwarding Resistance**: Be resilient to Forwarding and Replay Attacks. Having intercepted previous communications, an attacker can impersonate the legal user to authenticate to the system. The attacker can replay or forward the intercepted messages.

To prevent DoS attacks, the protocol must ensure that invalid information will not affect the user's authentication, which means the attacker cannot pass the authentication of RP by any means, thus making *CNTR* of RP and *CNTR* of the authenticator out of sync. Therefore, the protocol must ensure that every successful authentication of RP is associated with the legitimate user's verification. The Forgery Resistance, Parallel Session Resistance and Forwarding Resistance require the protocol to prevent impersonation attacks by replaying, constructing, or manipulating previous messages. Formally, RP must obtain injective agreement on *UName*, *AAID*, *KeyID*, *AppID* with the authenticator after the authentication process.

**SG-5 Verifier Leak Resilience:** Be resilient to leaks from other relying parties. I.e., nothing that a verifier could possibly leak can help an attacker impersonate the user to another relying party.

**SG-6 Authenticator Leak Resilience:** Be resilient to leaks from other FIDO Authenticators. I.e., nothing that a particular FIDO Authenticator could possibly leak can help an attacker to impersonate any other user to any relying party

We formalize them to: after the authentication process, RP must obtain injective agreement on *UName*, *AAID*, *KeyID*, *AppID* with the authenticator when another RP leaks the same user's $pk_{AU}$, *UName*, *AAID*, *KeyID*, *CNTR* or when another authenticator leaks the same user's $sk_{AU}$, *UName*, *AAID*, *CNTR*, *KeyID*.

Note that after a successful authentication for the user, it is the UA but not the user who has been authorized, if the correct user is verified but the malicious UA is authorized, the protocol is still not secure, so after the authentication process, RP must obtain injective agreement on *UName, AAID* with UA when another RP leaks the same user's $pk_{AU}$, *UName*, *AAID*, *KeyID*, *CNTR* and when another authenticator leaks the same user's $sk_{AU}$, *UName*, *AAID*, *CNTR*, *KeyID*.

In the registration process, the security reference only presents the following goal.

**SG-7 User Consent:** Notify the user before a relationship to a new relying party is being established (requiring explicit consent).

This goal indicates that the registration request must have been initiated by the legitimate user. We assume that the legitimate UA that initiated the registration request can represent the consent of the user, so RP must obtain injective agreement on *UName*, *AppID* with UA after the registration process.

However, even the registration process has been consented by the user, we cannot guarantee that it is the user's authenticator who has been registered. So the registration should additionally imply that RP must obtain injective agreement on *UName*, *AAID*, *KeyID*, *AppID*, $pk_{AU}$ with the authenticator after the registration process.

In a transaction confirmation process, any tampering of the transaction message during its route to the end device display and back should be detected and the user cannot deny the transaction message, which FIDO presents in SG-14.

**SG-14 Transaction Non-Repudiation:** Provide strong cryptographic non-repudiation for secure transactions.

Formally, RP must obtain injective agreement on *Tr* with the authenticator after the transaction confirmation process.

*2) Confidentiality Properties:* The confidentiality of $sk_{AT}$, $sk_{AU}$ and $k_W$ is required in Section 4.1 of the security reference [6]. Formally, the cryptographic key $sk_{AT}$, $sk_{AU}$ and $k_W$ should remain secret in the presence of the active attacker during the registration and the authentication process.

*KHAccessToken* ($ak$) is an access control mechanism for protecting an authenticator's UAF credentials from unauthorized use. Once $ak$ is leaked, the attacker will have the ability to impersonate ASM and call the authenticator. ASM is required to maintain the secrecy of $ak$ in Section 6.1 [9]. Formally, $ak$ should remain secret in the presence of the active attacker during the registration and the authentication process.

*3) Privacy Properties:* The UAF protocol should ensure that the private data related to the user cannot be compromised. Otherwise, the attacker can identify a user or trace a user's behavior.

First, *Tr* is sensitive data, so it must remain secret in the transaction confirmation process. Otherwise, the attacker can count transactions and track user behaviors. So, *Tr* should remain secret in the presence of the active attacker during the transaction confirmation process.

Similarly, *CNTR* must remain secret. Or, the number of successful authentications is known to the attacker, and the attacker can track user behaviors. So *CNTR* should remain secret in the presence of the active attacker during the registration and the authentication process.

The higher requirement for privacy is unlinkability, which is stated by:

**SG-4 unlinkability:** Protect the protocol conversation such that any two Relying Parties cannot link the conversation to one user.

The main purpose of this goal is to mitigate the potential for collusion amongst RPs. Generally, we disregard the linkability due to the irresistible external factors (same *UName*, same

IP address, etc.). So the UAF should provide unlinkability between different RPs.

## IV. Modeling UAF Protocol in ProVerif

In this section, we briefly introduce the formal verification tool ProVerif and explain how security properties can be modeled in ProVerif in general. Then, we present the modeling of security goals and protocol operations. For some of the attack models that are not integrated with ProVerif, we present the tricks we used to implement them. We believe other formal verification tools, such as Tamarin [46], AVISPA [16], can also be used to model and verify UAF. We choose ProVerif due to its popularity and ease of use.

### A. Overview of ProVerif

ProVerif is an automatic symbolic protocol verifier, which can verify various security properties of protocols, including confidentiality, authentication, and observational equivalence [23]. Comparing with other verification tools, such as AVISPA [16], CL-AtSe [51], OFMC [19], Tamarin [46], FDR [44], Scyther [31], SATMC [17], Cryptyc [35], TA4SP [25], Maude-NPA [34], ProVerif not only solves the problem of state explosion but also supports unbounded sessions. Although ProVerif does not support algebraic operations, such as power operation and XOR operation, it can still be used to verify the UAF protocol since the protocol does not use such operations.

ProVerif verifies in an extension of the applied $\pi$-calculus with cryptography. Based on first-order logic resolution rules on Horn clauses [53], it determines whether the desired security properties are met. If a property is violated, it can construct an attack at both the Horn clause level and the $\pi$-calculus level.

In $\pi$-calculus, messages are described as terms, and a term is constructed by constructors. For example, we can define `senc(M,K)` as a symmetric encryption of the message `M` under the key `K`, where `senc(bitstring, key)` is a constructor. The corresponding destructor `sdec(bitstring, key)` is defined as follows to formally represent the decryption of the ciphertext with the same key `K`:

```
fun senc(bitstring,key):bitstring.
reduc forall m: bitstring, k:key; sdec(senc(m,k),k)
    = m.
```

### B. Formalizing Security Goals in ProVerif

ProVerif can prove reachability properties, correspondence assertions, and equivalence properties.

Confidentiality is a reachability property. ProVerif checks all possible protocol executions and all possible attacker behaviors to infer which terms are available to the attacker. Using the following query statements, ProVerif tests the confidentiality of the term `M` and the confidentiality of the bound name or variable `x`.

```
query attacker(M).
query secret x.
```

Authentication properties can be verified via the *Correspondence assertions*. Correspondence assertions are used to capture the relationships between events. If the specified events can be executed in the correct order and they have the same arguments, the related properties can be guaranteed. For example, if entity A executes an event `e1` (A terminates the protocol with B) with the argument $x$ (B's identity) and there is an entity B that has executed an event `e2` (B started a session of the protocol with A) with the same argument $x$, from A's point of view, B has finished a non-injective agreement with A on data $x$. We can use the following query to check the non-injective agreement on data $x$:

```
query x:ID;
event(e1(x)) ==> event(e2(x)).
```

Unlinkability is an equivalence property, which could be verified using observational equivalence [23]. If the attacker cannot distinguish a process $P$ from a process $Q$, $P$ and $Q$ are observational equivalent $P \approx Q$ where the processes $P$ and $Q$ have the same structure and differ only in the choice of terms. In ProVerif, the equivalence is written by a single "biprocess", which encodes both $P$ and $Q$. Such a biprocess uses the construct `diff[M,M′]` to represent the terms that differ between $P$ and $Q$ (keyword `choice` is a synonym for `diff`), where $P$ uses the first component of the choice `M`, while $Q$ uses the second one `M′`. For example, if $P$ and $Q$ are protocols that have the same structure but only differ in the parameter $a$ ($P$ for $a_1$ and $Q$ for $a_2$), then the equivalence of $P$ and $Q$ can be expressed by: $P(a_1) \approx P(a_2)$. The processes can be expressed as follows, and ProVerif verifies whether they are equivalent.

```
free a:bitstring.
free b:bitstring.
let P_and_Q(M:bitstring) = ..... (* definition of
    the processes *)
process
    !P_and_Q(choice[a,b])
```

*Challenge.* Modeling the unlinkability goal in the UAF is difficult because ProVerif could only verify the observational equivalence from the perspective of the attacker. But the unlinkability requirement in UAF is from the perspective of RP. To model this situation, we need to make sure the attacker knows what RP knows. However, in our model, the attacker can participate in the protocol as malicious entities, and can actively manipulate the session data to break the security goals, where RP never does. So we model the unlinkability in the following way: i) when analyzing the unlinkability, we assume there are no malicious entities in the protocol run; ii) we assume the channel between RP and UA is public, which allows the attacker to have the same knowledge as RP; iii) we assume the attacker is a passive attacker who could only listen to the communication channel between RP and UA.

### C. ProVerif Models of the UAF

We modeled different types of authenticators and application scenarios in ProVerif, which takes 900 lines of ProVerif code. Then we analyzed whether the UAF protocol meets the security goals in different scenarios using different security assumptions and process combinations. We discuss the challenges we overcame:

*1) Modeling Malicious Entity Scenarios:* ProVerif models two types of channels, the *Public channel* and the *Private channel*. The public channel is assumed to be completely controlled by the attacker who has the "Dolev-Yao" capabilities, while the data on the private channel is excluded from the attacker's knowledge. Unfortunately, none of these channels can be directly used to model the malicious entities scenarios where the communication between honest entities is assumed secure but the attacker can act as a malicious entity to communicate with some of the entities.

To this end, we used a modeling trick and verified its correctness with the developers of ProVerif. We define two entity processes `A` and `A'` (both of them communicate with process `B`) running in parallel, where `A` communicates via a private channel while `A'` communicates via a public channel. The attacker can control the public channel, so he/she can act as a malicious `B` and communicate with `A'`. When removing the process `A'`, there is no malicious `B` in the environment.

This model becomes more complex in UAF because it contains more than two entities. Take ASM as an example, it communicates with two entities, UC and the authenticator. We define a process macro `ASM`, which contains two arguments of type *channel* `MC` and `MA`, where `MC` for communicating with UC and `MA` for communicating with the authenticator, as defined below:

```
let ASM(MC:channel, MA:channel) = (...)
```

By controlling the type of `MA` and `MC`, different scenarios can be modeled. The following statements represent that from ASM's point of view, the system has no malicious authenticator and UC since there is only an ASM that uses private channel `MC` and `MA` to communicate. Note that channels defined in the process by "new" are private.

```
free c:channel[public].
process
    new MC:channel;
    new MA:channel;
    ASM(MC, MA) | UC(MC) | Authenticator(MA)
```

The following statements represent a system that has malicious authenticators since there is an ASM that uses public channel `c` to communicate with the authenticator.

```
free c:channel[public].
process
    new MC:channel;
    new MA:channel;
    ASM(MC, MA) | ASM(MC, c) | UC(MC) |
        Authenticator(MA)
```

This trick requires two processes (only differ in channels) to run in parallel. However, ProVerif 2.00 does not allow the same event to execute several times in the same branch in verifying event correspondence, which raises the difficulty for ProVerif to verify such scenarios.

To solve this challenge, we used an 'if' statement to force the process macro that contains the test events to only present once in a branch. As the following code shows, we run two ASM processes in parallel with different channel parameters to make sure only one of them is on a certain branch. The main process receives branch information on the public channel `c` and lets the attacker choose between the two branches. We confirmed the correctness of this trick with the developers of ProVerif, who also fixed the problem in ProVerif 2.01.

```
free c:channel[public].
process
!(
    new MA:channel;
    new MC:channel;
    Authenticator(MA)|
    (
        in(c,branch:bool);
        if branch = True then
            ASM(MC, MA)
        else
            ASM(MC, c)
    )
)
```

*2) Modeling Unlinkability Scenarios:* Modeling the UAF operations for unlinkability analysis is challenging. We explicitly model the scenario in which two RPs might authenticate the same user or different users.

First, we consider two RPs run in parallel with different *AppID*. Then, we define a `System` macro, which represents protocol runs that an RP authenticates or registers a user. By controlling the arguments of the `System`, we specify which RP is running. For example, `System(AppID1,...)` means RP with *AppID1* is running, while `System(AppID2,...)` means a different RP with *AppID2* is running. Last, we specify two `Systems` to run. One of them represents an RP that authenticates a user, whereas the other represents another RP that authenticates the same user or authenticates another user.

Different users use different devices, so their authenticators have different $k_W$, and their ASMs have different *tok*. Whether the two RPs authenticate the same user, $sk_{AU}$ and *KeyID* in the system is different because the authenticator always generates a new $sk_{AU}$ and *KeyID* for each account during the registration process. Since different users may use authenticators with the same *AAID*, we do not consider the differences of *AAID*. Similarly, in different user devices, *CallerID* and *FacetID* could be the same, so we do not consider either. We do not consider the impact of *UName* of the unlinkability.

```
process
    ...
    system(AppID, AAID, skAU, KeyID, kW, tok, UName,
         FacetID, CallerID) |
    system(AppID2, AAID, choice[skAU,skAU2], choice[
        KeyID,KeyID2], choice[kW,kW2], choice[tok,
        tok2], UName, FacetID, CallerID)
```

## V. Security Analysis

In this section, we present the formal verification results of the UAF protocol. We identify the minimal security assumptions required for each of the security goals in Section III-B to hold. The formal verification identifies the design flaws in the UAF, but not specific implementation vulnerabilities in different apps. The root causes of these flaws include that FIDO UAF supports different deployment settings but gives impractical and ambiguous security assumptions for such settings.

The results are analyzed from 417,792 automatically generated cases considering different authenticator types, scenarios, and assumptions. It took 80 hours to analyze all cases on a computer with Intel(R) Core(TM) i7-3770 CPU and 16GB RAM. Section V-A describes the method for automatically identifying minimal security assumptions. Section V-B presents the result overview. Section V-C presents some attacks. Section V-D describes our recommendations.

## A. Automatically Identifying Minimal Security Assumptions

To automatically identify minimal security assumptions, we developed a tool UAFVerif to generate cases corresponding to different scenarios for ProVerif to perform analysis on. UAFVerif is implemented in 680 lines of Python code. The high-level idea is to define a variable security assumption set $\mathcal{A}$, where $\mathcal{A} = \emptyset$ represents no security assumption. UAFVerif adds security assumptions $\{a_1, ..., a_n\}$ into $\mathcal{A}$ and triggers ProVerif to analyze if the protocol satisfies the security properties under these security assumptions. It starts with verifying a single security assumption, then all combinations of 2 assumptions, and so on. When the state of a security property changes from unmet to met, the minimal security requirement is recorded.

## B. Result Overview

| Reg. | Type | 1B | 2B | 1R | 2R |
|---|---|---|---|---|---|
| C. | $k_W$ | $\surd$ | | | |
| | $sk_{AT}$ | $\surd$ | | | |
| | $sk_{AU}$ | $\neg k_W \vee \neg M[A]$ | | $\surd$ | $\neg k_W$ |
| | $ak$ | $\neg tok \wedge \neg A[M]$ | | $\times$ | |
| | *CNTR* | $\neg C[M] \wedge \neg M[A]$ | | | |
| A. | Basic | $\neg C[U] \wedge \neg M[C] \wedge \neg A[M]$ | | | |

TABLE III. MINIMAL ASSUMPTIONS REQUIRED FOR THE UAF REGISTRATION PROCESS TO ACHIEVE CONFIDENTIALITY PROPERTIES AND AUTHENTICATION PROPERTIES.

Table III and IV present the minimal assumptions required for UAF to achieve the confidentiality properties and authentication properties. 'Reg.' means the registration process, 'Auth.' means the authentication process. 'C.' means the confidentiality properties. 'A.' means the authentication properties. 'Basic' represents the authentication goals we explain in Section III-B, 'Non-R' represents the transaction non-repudiation goal. We present security assumptions in symbols: '$\wedge$' denotes AND, whereas '$\vee$' denotes OR. *A, M, C, U* represent Authenticator, ASM, UC, UA, respectively. '$\neg$' before a data field represents the field is not compromised. '$\neg$' before '$X[Y]$' represents that the system does not exist malicious $X$ that can communicate with $Y$. For example, '$\neg M[A]$' means that there is no malicious ASM that can communicate with the authenticator, while '$\neg M[C]$' means that there is no malicious ASM in the system that can communicate with UC. '$\surd$' means the protocol meets the security goal under all conditions. '$\times$' means the protocol cannot meet the security goal nonetheless. '$-$' means we do not consider this property.

*1) Confidentiality Properties:* As Table III shows, the protocol does not disclose $k_W$ or $sk_{AT}$ in the registration process because they do not leave the authenticator. However, $h$, which is encrypted by $k_W$ and contains $sk_{AU}$, will leave the authenticator (except for 1R authenticator, which stores $h$ inside its internal storage). Therefore, if $k_W$ is compromised, the attacker can decrypt $h$ and get $sk_{AU}$. For 1B and 2B authenticators, $h$ is only sent from the authenticator to ASM. So as long as there is no malicious ASM, the attacker cannot obtain $sk_{AU}$. Confidentiality of $sk_{AU}$ holds when using 1R authenticators since it never leaves the authenticator. 2R authenticators take $h$ as *KeyID* and send it to RPs, therefore, the protocol should guarantee that $k_W$ will not be compromised. As Table IV shows in the authentication process, for 1B and 2B authenticators, if $k_W$ is secure, the attacker cannot decrypt $h$ to get $sk_{AU}$. If $k_W$ is compromised, the attacker cannot obtain $h$ from ASM assuming no malicious authenticators. For 1R authenticators, since $h$ does not leave the authenticator, as long as the authenticator is not compromised, $sk_{AU}$ is secure. For 2R authenticators, since $h$ will be sent from RP, the minimal assumption is to keep the confidentiality of $k_W$.

To maintain the confidentiality of $ak$ in 1B and 2B authenticators, $tok$ cannot be compromised and there should not be a malicious authenticator. If the attacker gets $tok$, he/she can compute $ak = \texttt{hash}(\textit{AppID} \,||\, \textit{tok} \,||\, \textit{CallerID})$. Whether it is the registration process or authentication process, $ak$ needs to be sent to the authenticator by ASM. If there is a malicious authenticator, the attacker can get the message, which contains $ak$, from ASM. When using 1R and 2R authenticators, $ak$'s confidentiality cannot be satisfied nonetheless, because $ak$ only contains *AppID*, which is public and known by the attacker. Therefore, *the KHAccessToken mechanism is futile*. We will discuss attacks on this issue in Section V-C and provide a fix in Section V-D2.

To maintain the confidentiality of *CNTR* in the registration process, the deployment of the protocol must satisfy $\neg C[M]$ and $\neg M[A]$. Otherwise, the malicious entity can initiate a registration process to get *CNTR* generated by the authenticator. In the authentication process, for 1B and 2B authenticators, to maintain the confidentiality of *CNTR*, the protocol must satisfy $\neg k_W$ and $\neg A[M]$, or satisfy $\neg M[A]$. Since ASM has access control over UC's *CallerID*, UC checks *FacetID* of the UA, so malicious UCs and UAs cannot start the legitimate authenticator. So when there is no malicious ASM, the confidentiality of *CNTR* holds. Besides, ASM needs to send $h$ to start the authenticator in 1B and 2B authenticators. If there is no malicious authenticator and $k_W$ is not compromised, the attacker cannot get $h$, even there is a malicious ASM, the attacker cannot provide $h$ and start the legitimate authenticator, the confidentiality of *CNTR* holds. For 1R and 2R authenticators, ASM does not verify UC's *CallerID*. A malicious UC can call ASM and get *CNTR* from the authenticator. So the deployment of the protocol should ensure there is no malicious UC and there is no malicious ASM.

Any malicious UC, ASM, or authenticator can get *Tr* from the caller. This is because *Tr* is sent from RP to the authenticator, UA does not verify UC, UC does not verify ASM, and ASM does not verify authenticator in the authentication process.

| Auth. | Type | 1B |  | 2B | 1R |  | 2R |
|---|---|---|---|---|---|---|---|
|  |  | login | step-up | step-up | login | step-up | step-up |
| C. | $sk_{AU}$ | $\neg k_W \vee \neg A[M]$ |  | $\neg k_W \vee \neg A[M]$ | $\surd$ |  | $\neg k_W$ |
|  | $ak$ | $\neg tok \wedge \neg A[M]$ |  | $\neg tok \wedge \neg A[M]$ | $\times$ |  | $\times$ |
|  | *CNTR* | $(\neg k_W \wedge \neg A[M]) \vee \neg M[A]$ |  | $(\neg k_W \wedge \neg A[M]) \vee \neg M[A]$ | $\neg C[M] \wedge \neg M[A]$ |  | $\neg C[M] \wedge \neg M[A]$ |
|  | *Tr* | – | $\neg C[U] \wedge \neg M[C] \wedge \neg A[M]$ | $\neg C[U] \wedge \neg M[C] \wedge \neg A[M]$ | – | $\neg C[U] \wedge \neg M[C] \wedge \neg A[M]$ | $\neg C[U] \wedge \neg M[C] \wedge \neg A[M]$ |
| A. | Basic | $\neg A[M] \vee \neg M[A]$ | $\surd$ | $\surd$ | $\neg C[M] \wedge \neg M[A]$ | $\surd$ | $\surd$ |
|  | Non-R | – | $\surd$ | $\surd$ | – | $\surd$ | $\surd$ |

TABLE IV. MINIMAL ASSUMPTIONS REQUIRED FOR THE UAF AUTHENTICATION PROCESS TO ACHIEVE CONFIDENTIALITY PROPERTIES AND AUTHENTICATION PROPERTIES.

*2) Authentication Properties:* As shown in Table III, to achieve the authentication goals in registration, the minimal assumption is $\neg C[U]$ and $\neg M[C]$ and $\neg A[M]$. The results show that whether $sk_{AT}$ is compromised has little influence on the authentication goals because the attacker has an authenticator with the same $sk_{AT}$ to register. Therefore, $sk_{AT}$ can only guarantee that the authenticator registered in RP must be legitimate, but cannot guarantee the authentication goals of the registration process. To achieve the authentication goals in the registration process, the protocol should guarantee that it is the user's authenticator that is bound to his/her account but not the attacker's authenticator, which requires $\neg C[U]$ and $\neg M[C]$ and $\neg A[M]$. ProVerif generated an attack when one of them is not satisfied, which we will discuss in Section V-C. The root cause of this issue is because there is no access control mechanism from UA to UC, from UC to ASM, or from ASM to the authenticator in the registration process. For example, a UA may send the UAF request to any UC installed on the user's devices, including a malicious one.

The confidentiality of $sk_{AU}$, $k_W$, $tok$, and $CNTR$ is crucial for authentication properties in the authentication process. $k_W$ protects $sk_{AU}$. When the attacker get $k_W$, he/she can decrypt $h$ and get $sk_{AU}$. When $tok$ is compromised, the attacker can construct $ak$ and use a malicious ASM to start the authenticator. The confidentiality of $sk_{AU}$ and *CNTR* ensures the authentication and non-repudiation goals. As long as one of them does not leak, the attacker cannot undermine authentication properties. For simplicity, these results are not shown in Table IV.

During the authentication process, the minimal assumptions vary under different types of authenticators and use cases. The protocol satisfies the authentication goals in step-up authentication phases since it is based on a successful login authentication. In the login case, for 1B authenticators, the protocol must satisfy $\neg A[M]$ or $\neg M[A]$. Otherwise, the attacker can get $ak$ by a malicious authenticator and pass the KHAccessToken mechanism to start the authenticator. For 1R authenticators, ASM does not verify *CallerID* of UC, so the minimal assumptions additionally require $\neg C[M]$. Besides, for 1R authenticators, the attacker knows $ak$, which means he/she can pass the KHAccessToken mechanism anytime. Therefore, to guarantee the authentication goals, the protocol must satisfy $\neg M[A]$, and the minimal assumption comes into $\neg C[M]$ and $\neg M[A]$.

In addition, the results show the protocol holds non-repudiation on *Tr* as long as $k_W$, $sk_{AU}$, $tok$ are not compromised.

The results show that the protocol can prevent attacks from malicious RP and UA because UC verifies *FacetID*. For example, when UA is a browser and the user visits a phishing site, the *FacetID*, which is the malicious web origin, will not pass the verification of UC. If UA is an application, only when UA is compromised or the user uses a malicious UA, it would visit a malicious RP. In this situation, *FacetID*, which is the identifier of the malicious UA, is not in the trusted user agent list retrieved from *AppID*.

*3) Unlinkability Property:* The verification results show that the protocol can satisfy unlinkability in both the registration process and the authentication process. In the registration process, it is because different users can register using the authenticators with the same *AAID* and $sk_{AT}$. RPs cannot confirm that the two registrations are conducted by the same user through *AAID* or $pk_{AT}$, nor can they confirm that the two registrations are conducted by different people according to *AAID* or $pk_{AT}$.

Regardless of whether two accounts registered on two RPs are from the same user, the authenticator has generated different $sk_{AU}$ and *KeyID* during the registration process, so the two RPs cannot distinguish whether the verified accounts are from the same user based on $pk_{AU}$ or *KeyID*. In fact, the information that RP can obtain during an authentication process includes *UName*, *AAID*, *CallerID*, *FacetID*, *KeyID*, *CNTR*. *AAID* cannot be used to distinguish two users since a large number of authenticators share the same *AAID*. *CallerID* and *FacetID* are identifiers related to the applications and the platforms, which cannot be used to identify the user either. RPs cannot collude and use *CNTR*s to find out the accounts of the same user since *CNTR*s are independent. So none of the fields helps RPs to distinguish whether the accounts come from the same user, the protocol holds the unlinkability in the authentication process.

## C. Attacks

Guided by the analysis results presented in V-B, we discover 4 types of attacks: i) authenticator rebinding attack, in which the attacker binds his/her authenticator to the victim's account; ii) parallel session attack, in which the attacker impersonates the victim by using the victim's authenticator to sign requests; iii) privacy disclosure attack, in which the victim's *Tr* and *CNTR* will be leaked; iv) denial of service attack, in which the attacker can invalidate the victim's legitimate authenticator.

*1) Authenticator Rebinding Attack:* When the deployment of the protocol does not meet the assumptions $\neg C[U]$, $\neg M[C]$ or $\neg A[M]$, the injective agreement on *UName*, *AAID*, *KeyID*, *AppID* between RP and the authenticator in the registration process cannot be satisfied, rebinding attacks can be performed. To do so, the attacker needs to convince the user to install a malicious UC, ASM, or authenticator into his/her device.

The attack has the following steps: i) the victim uses UA to log in to RP in the traditional way and initiate the UAF registration; ii) UA sends the registration request to the malicious UC. Or, UC sends the request to the malicious ASM. Or, ASM sends the request to the malicious authenticator; iii) the malicious UC redirects the request to the attacker's device; iv) the attacker uses his/her authenticator to continue the UAF operations with the redirected request; v) the attacker sends the response message to the malicious UC on the victim's device and forwards it to RP via UA; vi) the attacker completes the UAF registration on behalf of the victim and successfully rebinds the victim's identity to the attacker's authenticator. As a result, the attacker can bypass the authentication of RP and impersonate the victim.

To verify the feasibility of this attack on real-world apps, we compiled a dataset of 1,856 payment-related Android applications and identified 42 that use the UAF protocol. These apps can be divided into two categories depending on the authenticator type. 8 out of the 42 apps have a hardware-based authenticator, e.g., China Mobile Pay, whereas the other 34 use a software-based authenticator, e.g., Jingdong Finance.

We successfully carried out authenticator rebinding attacks on China Mobile Pay and Jingdong Finance. The other 40 apps may also be vulnerable to these attacks. We chose China Mobile Pay and Jingdong Finance because of their popularity. As of October 2020, China Mobile Pay has 214,424,508 downloads in total, and Jingdong Finance has 1,043,164,317 downloads in total. In March 2020, China Mobile Pay has monthly active users of 3,838,000 and Jingdong Finance has monthly active users of 23,116,000.

We reported the vulnerability of China Mobile Pay to the China National Vulnerability Database of Information Security (CNNVD) on May 25, 2020, resulting in a medium-risk vulnerability ID (CNNVD-202005-1219) on July 31, 2020. We disclosed the vulnerability on Jingdong Finance to JD Security Response Center on December 12, 2018, who replied on December 19, 2018, stating they would ignore the vulnerability.

China Mobile Pay (package name: `com.cmcc.hebao`, version: 7.6.70, MD5: 384c99ecd3ac0ea0f805959da2b76608) in Android deploys the UAF protocol by calling a third-party UC and uses the hardware authenticator. However, the application (UA) does not authenticate the entity it calls, which means it may call any UC, including a malicious one (lacking the assumption of $\neg C[U]$). Therefore, once the user selects the malicious UC to call, the attacker can carry out the authenticator rebinding attack.

Similar attacks can be performed when the attacker compromises UA, UC, ASM, or authenticator, but this requires higher capabilities of the attacker. The mis-binding attack [36] is similar to the authenticator rebinding attack, except that the former requires the attacker to corrupt UC and ASM. Cloned authenticator attack [47] needs the attacker to evade the security mechanism of the environment of the authenticator, get information of the user's authenticator, and deploy a malicious cloned authenticator. We performed the attack on Jingdong Finance (package name: `com.jd.jrapp`, version: 5.0.1, MD5: d56a8c05ab61194251b00b873fa3d4c4) by hooking the ASM and redirected messages to the attacker's device.

*2) Parallel Session Attack:* Parallel session attacks can be performed during the authentication process when the deployment of the protocol does not satisfy the assumptions $\neg A[M]$ or $\neg M[A]$ for the 1B login case, or does not satisfy the assumptions $\neg C[M]$ and $\neg M[A]$ for the 1R login case. In other words, to carry out the attack in the 1B login case, there should be a malicious authenticator and a malicious ASM on the victim's device, whereas to perform the attack in the 1R login case there should be a malicious UC or a malicious ASM on the victim's device. A similar attack was proposed [36], which requires the compromise of the legitimate UC and ASM, which is harder to pull off.

For 1B authenticators, the attack can be carried out in 2 steps. In step 1, i) the victim tries to log in to RP; ii) the legitimate ASM forwards the request, which contains $ak$ and $h$, to the malicious authenticator. Since the malicious authenticator does not have $sk_{AU}$ and $CNTR$, it cannot generate a valid authentication response. In step 2: i) the victim tries to log in to RP again; ii) the attacker sends a login request to RP with the victim's account at the same time and gets *Chlg*; iii) the malicious ASM sends the request message with the valid $ak$, $h$ and the attacker's *Chlg* to the legitimate authenticator; iv) unbeknownst to the victim, the victim verifies the fingerprint, and the authenticator signs the message and generates the authentication response.

Similarly, for 1R authenticators, the attack can be carried out in 2 steps. In step 1, the attacker computes $ak = \texttt{hash}(\mathit{AppID})$. In step 2: i) the victim tries to log in to RP; ii) the attacker sends a login request to RP with the victim's account at the same time and gets *Chlg*; iii) the malicious ASM sends the request message with the valid $ak$ and the attacker's *Chlg* to the legitimate authenticator, or the malicious UC sends the request message with the attacker's *Chlg* to the legitimate ASM; v) unbeknownst to the victim, the victim verifies the fingerprint, and the authenticator signs the message and generates the authentication response.

*3) Privacy Disclosure Attack:* When the deployment of the protocol does not satisfy the assumptions $\neg C[U]$ or $\neg C[M]$, some of the user's personal data will be leaked. Assuming there is a malicious UC and using the 1B authenticator: i) the victim tries to log in to RP or make a transaction with RP; ii) the malicious UC receives authentication request; iii) the attacker can confirm whether the victim is logging in or making a transaction depending on whether *Tr* is included in the request.

For 1R and 2R authenticators, ASM does not verify UC's *CallerID*: i) the victim tries to log in to RP or make a transaction with RP; ii) the malicious UC receives the authentication request, which may contain *Tr*; iii) the malicious UC sends the authentication request to the legitimate ASM, and the legitimate authenticator signs and performs other operations; iv) the malicious UC gets the authenticator response, which contains *CNTR*, signature, etc.; v) malicious UC forwards authentication response, the authentication succeeds. The attacker can use *CNTR* to compute how many times the victim tries to log in or make transactions. The attacker can also use the signature for chosen-ciphertext attacks.

*4) Denial of Service Attack:* The attacker can carry out a DoS attack when the deployment of the protocol does not satisfy the assumptions $\neg C[U]$ or $\neg M[C]$ or $\neg A[M]$. For all 4 types of authenticators, if there is a malicious UC, ASM, or authenticator, the attacker can discard the authentication request and the user cannot finish the authentication.

For 1R and 2R authenticators, if there is a malicious UC or ASM, the attacker can use the following steps to permanently disable the authenticator by making *CNTR* of the legitimate authenticator out of sync with RP: i) the victim tries to log in to RP or make a transaction with RP; ii) the malicious UC receives the authentication request from legitimate UA, forwards the request to the authenticator via ASM, and the authenticator signs. The authenticator's *CNTR* increments accordingly; iii) when the authentication response is returned to the malicious UC, it intercepts the request and sends a fail message to UA; iv) RP gets a failure, so *CNTR* does not increment; v) the attacker can repeat this attack multiple times to cause *CNTR* out of sync. Note that the UAF protocol does not have a *CNTR* synchronization mechanism.

## D. Recommendations

We present several concrete recommendations to enhance the security of the UAF protocol.

*1) Explicit Requirements:* The security goals given by the security reference are informal and fragmentary: i) SG-1 is ambiguous in which no clear definition of 'strong authentication' is provided; ii) except SG-1, all other authentication properties were presented in the format of resilience to some known attacks or vulnerabilities, which do not evolve as new attacks are discovered; iii) only the user consent goal is explicitly presented for the registration process, which has several implicit goals as we discussed in Section III-B.

We recommend presenting security requirements and goals in a more explicit and active way in the specifications. For example, the specifications can use formal expressions to describe the security properties as shown in Section III-B. The analysis results show that the registration process is more vulnerable to attack, so the specifications should improve the description of authentication properties for the registration process.

*2) Modify the KHAccessToken Mechanism:* The analysis results show that the KHAccessToken mechanism is futile to prevent the malicious ASMs: i) the confidentiality of $ak$ cannot be held when using 1R and 2R authenticators; ii) the confidentiality of $ak$ cannot be held when there is a malicious authenticator. The authenticator's trust on ASM is based on the Trust On First Use (TOFU) concept [10], which means it assumes there is no malicious ASM in the registration process but there can be malicious ASMs in the authentication process. However, it is equally difficult for the attacker to trick the victim into installing a malicious ASM in the registration process or the authentication process. So this mechanism is futile to prevent malicious ASMs. Furthermore, even if ASM is trusted in the registration process, the attacker can still get $ak$ from ASM by receiving authentication requests with malicious authenticators.

We give the following recommendations: i) there should be requirements for mechanisms to guarantee the security of the running environment of ASM and the authenticator. And, vendors must ensure the deployment of the protocol satisfies $\neg A[M]$ and $\neg M[A]$, e.g., running ASM and the authenticator in a trusted execution environment; ii) the KHAccessToken mechanism for 1R and 2R authenticators should be improved to prevent malicious ASM from communicating with the authenticator. For example, same as bound authenticators, $ak$ should include $tok$. To this end, the authenticator can maintain a trusted list of ASMs; iii) there should be a mechanism for the authenticator to authenticate ASM. For example, in 1B and 2B authenticators, vendors can provide a shared key in both ASM and the authenticator, so the communication between ASM and the authenticator can be encrypted.

*3) Authenticating UC at ASM:* If the deployment of the protocol does not satisfy $\neg C[M]$, some security properties will not hold. Although Section 6.2 of ASM specification states that ASM must implement the access control of *CallerID* [9], it does not specify how to verify *CallerID* and leave the implementation of the security mechanisms to the vendors. We emphasize the importance of this issue and suggest to have a standard ASM access control mechanism for *CallerID* in the specification. For example, ASM can maintain a trusted *CallerID* list. Only UCs with a valid *CallerID* can communicate with ASM.

*4) Authenticating UC at UA:* The protocol does not require UA to authenticate UC. As a result, UA or the user may invoke a malicious UC installed on the device. We recommend the specifications to require UA to authenticate UC. For example, UA can use the same mechanism as ASM does to authenticate UC.

## VI. Related Work

Hu et al. manually abstracted the UAF protocol and presented 3 attacks, including mis-binding attack, parallel session attack, and multi-user attack [36]. Leoutsarakos manually found 15 defects of the UAF protocol, which were not formally verified [42]. Panos et al. presented a manual and informal analysis of the UAF protocol with several discovered vulnerabilities and attacks [47]. Loutfi et al. gave a set of trust requirements of the FIDO UAF protocol which included the trust requirements in FIDO consortium, in service providers, in a hardware manufacturer, in a local device computing platform, and in the end-user [43]. Zhang et al. presented an attack on FIDO transaction confirmation and proposed a secure display mechanism for mobile devices [56].

Jacomme et al. found that the U2F protocol can guarantee authentication in many threat scenarios, such as with phishing sites, but cannot achieve authentication goals in the presence of malware in the user environment [38]. Although the paper analyzed different scenarios, no formal model was given. Meanwhile, the model description of the U2F protocol was simple, and it did not consider different security assumptions and optional protocol operations. Chang et al. found that the U2F protocol could leak two fixed keys (attestation key and device secret key) through a side-channel attack. They recommended a modification of the U2F protocol to minimize the effect of this attack and presented a new variant of the U2F protocol to provide a stronger security guarantee. Similarly, they introduced how to perform side-channel attacks on the UAF protocol [29]. Pereira et al. formally analyzed the authentication properties of the U2F protocol [48]. They analyzed two types of U2F clients with and without *AppID* verification and found that the U2F protocol could not satisfy authentication without *AppID* verification. However, the protocol model was oversimplified.

Different from previous work, we provide a faithful formalization of the UAF protocol and use the formal method to analyze the UAF protocol.

## VII. Conclusion

In this paper, we formally analyzed the UAF protocol. We formalized the security assumptions and goals of the protocol, provided a formal model of the protocol, and used ProVerif to analyze the protocol under different scenarios. Our analysis identified the minimal security assumptions required for each of the security goals of the UAF protocol. By summarizing and analyzing the results given by ProVerif, we presented the defects of the protocol and illustrated some attacks. We also confirmed previously discovered vulnerabilities in an automated way and disclosed several new attacks. Also, we offered several concrete recommendations to fix the identified problems and weaknesses in the protocol.

## Acknowledgment

We thank Malihek Shirvanian and the anonymous reviewers for their comments. The research of Beijing University and Posts and Telecommunications is partially funded by the Technology Major Project of China (2018ZX03001010-005).

## References

[1] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta *et al.*, “Imperfect forward secrecy: How diffie-hellman fails in practice,” in *ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 5–17.

[2] F. Alliance, “Android Now FIDO2 Certified, Accelerating Global Migration Beyond Passwords,” https://fidoalliance.org/android-now-fido2-certified-accelerating-global-migration-beyond-passwords/, 2017.

[3] ——, “Client to authenticator protocol (ctap),” https://fidoalliance.org/specs/fido-v2.0-ps-20190130/fido-client-to-authenticator-protocol-v2.0-ps-20190130.html, 2017.

[4] ——, “Fido appid and facet specification,” https://fidoalliance.org/specs/fido-uaf-v1.1-ps-20170202/fido-appid-and-facets-v1.1-ps-20170202.html, 2017.

[5] ——, “Fido members,” https://fidoalliance.org/members/, 2017.

[6] ——, “Fido security reference,” https://fidoalliance.org/specs/fido-uaf-v1.1-ps-20170202/fido-security-ref-v1.1-ps-20170202.html, 2017.

[7] ——, “Fido uaf architectural overview,” https://fidoalliance.org/specs/fido-uaf-v1.1-ps-20170202/fido-uaf-overview-v1.1-ps-20170202.html, 2017.

[8] ——, “Fido uaf authenticator commands,” https://fidoalliance.org/specs/fido-uaf-v1.1-ps-20170202/fido-uaf-authnr-cmds-v1.1-ps-20170202.html, 2017.

[9] ——, “Fido uaf authenticator-specific module api,” https://fidoalliance.org/specs/fido-uaf-v1.1-ps-20170202/fido-uaf-asm-api-v1.1-ps-20170202.html, 2017.

[10] ——, “Fido uaf protocol specification,” https://fidoalliance.org/specs/fido-uaf-v1.1-ps-20170202/fido-uaf-protocol-v1.1-ps-20170202.html, 2017.

[11] ——, “Fido authenticator allowed cryptography list,” https://fidoalliance.org/specs/fido-security-requirements-v1.2-2018/fido-authenticator-allowed-cryptography-list-v1.0-wd-20180629.html, 2018.

[12] ——, “Fido authenticator allowed restricted operating environments list,” https://fidoalliance.org/specs/fido-security-requirements-v1.2-2018/fido-authenticator-allowed-restricted-operating-environments-list-v1.0-wd-20180629.html, 2018.

[13] ——, “Fido authenticator metadata requirements,” https://fidoalliance.org/specs/fido-security-requirements-v1.2-2018/fido-authenticator-metadata-requirements-v1.0-wd-20180629.html, 2018.

[14] ——, “Fido authenticator security requirements,” https://fidoalliance.org/specs/fido-security-requirements/fido-authenticator-security-requirements-v1.3-fd-20180905.html, 2018.

[15] ——, “FIDO Certified Products,” https://fidoalliance.org/certification/fido-certified-products/, 2019.

[16] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. H. Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani *et al.*, “The avispa tool for the automated validation of internet security protocols and applications,” in *International conference on computer aided verification*. Springer, 2005, pp. 281–285.

[17] A. Armando, R. Carbone, and L. Compagna, “Satmc: a sat-based model checker for security-critical systems,” in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2014, pp. 31–45.

[18] D. Basin, J. Dreier, L. Hirschi, S. Radomirovic, R. Sasse, and V. Stettler, “A formal analysis of 5g authentication,” in *ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1383–1396.

[19] D. Basin, S. Mödersheim, and L. Vigano, “Ofmc: A symbolic model checker for security protocols,” *International Journal of Information Security*, vol. 4, no. 3, pp. 181–208, 2005.

[20] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. Zinzindohoue, “A messy state of the union: Taming the composite state machines of tls,” in *IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 535–552.

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

[22] K. Bhargavan, C. Fournet, R. Corin, and E. Zalinescu, “Cryptographically verified implementations for tls,” in *ACM conference on Computer and communications security*. ACM, 2008, pp. 459–468.

[23] B. Blanchet, “Modeling and verifying security protocols with the applied pi calculus and proverif,” vol. 1, no. 1-2, pp. 1–135, 2016.

[24] B. Blanchet, B. Smyth, V. Cheval, and M. Sylvestre, *ProVerif 2.00: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial*, May 2018, http://prosecco.gforge.inria.fr/personal/bblanche/proverif/manual.pdf.

[25] Y. Boichut, N. Kosmatov, and L. Vigneron, “Validation of prouvé protocols using the automatic tool ta4sp,” in *Taiwanese-French Conference on Information Technology (TFIT 2006)*, 2006, pp. 467–480.

[26] J. Bonneau, “The science of guessing: analyzing an anonymized corpus of 70 million passwords,” in *IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 538–552.

[27] J. Bonneau, C. Herley, P. C. Van Oorschot, and F. Stajano, “The quest to replace passwords: A framework for comparative evaluation of web authentication schemes,” in *IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 553–567.

[28] D. Chang, X. Chu, and G. Wei, “Analysis of the security-enhanced vtpm migration protocol based on proverif,” in *International Conference on Computational and Information Sciences*. IEEE, 2013, pp. 1437–1440.

[29] D. Chang, S. Mishra, S. K. Sanadhya, and A. P. Singh, “On making u2f protocol leakage-resilient via re-keying.” *IACR Cryptology ePrint Archive*, vol. 2017, p. 721, 2017.

[30] C. Cremers, “Key exchange in ipsec revisited: Formal analysis of ikev1 and ikev2,” in *European Symposium on Research in Computer Security*. Springer, 2011, pp. 315–334.

[31] C. J. Cremers, “Unbounded verification, falsification, and characterization of security protocols by pattern refinement,” in *ACM conference on Computer and communications security*. ACM, 2008, pp. 119–128.

[32] S. Delaune, S. Kremer, and M. Ryan, "Verifying privacy-type properties of electronic voting protocols," *Journal of Computer Security*, vol. 17, no. 4, pp. 435–487, 2009.

[33] N. Dong, H. Jonker, and J. Pang, "Formal analysis of an e-health protocol," *arXiv preprint arXiv:1808.08403*, 2018.

[34] S. Escobar, C. Meadows, and J. Meseguer, "A rewriting-based inference system for the nrl protocol analyzer and its meta-logical properties," *Theoretical Computer Science*, vol. 367, no. 1-2, pp. 162–202, 2006.

[35] A. D. Gordon and A. Jeffrey, "Types and effects for asymmetric cryptographic protocols," *Journal of Computer Security*, vol. 12, no. 3-4, pp. 435–483, 2004.

[36] K. Hu and Z. Zhang, "Security analysis of an attractive online authentication standard: Fido uaf protocol," *China Communications*, vol. 13, no. 12, pp. 189–198, 2016.

[37] S. Islam, "Security analysis of lmap using avispa," *International journal of security and networks*, vol. 9, no. 1, pp. 30–39, 2014.

[38] C. Jacomme and S. Kremer, "An extensive formal analysis of multi-factor authentication protocols," in *IEEE Computer Security Foundations Symposium (CSF)*. IEEE, 2018, pp. 1–15.

[39] N. Karapanos and S. Capkun, "On the effective prevention of tls man-in-the-middle attacks in web applications," in *Usenix Security Symposium*, 2014, pp. 671–686.

[40] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual international cryptology conference*. Springer, 1999, pp. 388–397.

[41] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Annual International Cryptology Conference*. Springer, 1996, pp. 104–113.

[42] N. Leoutsarakos, "What's wrong with fido?" http://zeropasswords.com/pdfs/WHATisWRONG_FIDO.pdf, 2011.

[43] I. Loutfi and A. Jøsang, "Fido trust requirements," in *Nordic Conference on Secure IT Systems*. Springer, 2015, pp. 139–155.

[44] G. Lowe, "Breaking and fixing the needham-schroeder public-key protocol using fdr," in *International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 1996, pp. 147–166.

[45] ——, "A hierarchy of authentication specifications," in *Computer Security Foundations Workshop*. IEEE, 1997, pp. 31–43.

[46] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The tamarin prover for the symbolic analysis of security protocols," in *International Conference on Computer Aided Verification*. Springer, 2013, pp. 696–701.

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

[48] O. Pereira, F. Rochet, and C. Wiedling, "Formal analysis of the fido 1.x protocol," in *International Symposium on Foundations & Practice of Security*, 2017.

[49] B. Schmidt, S. Meier, C. Cremers, and D. Basin, "Automated analysis of diffie-hellman protocols and advanced security properties," in *IEEE Computer Security Foundations Symposium*. IEEE, 2012, pp. 78–94.

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

[51] M. Turuani, "The cl-atse protocol analyser," in *International Conference on Rewriting Techniques and Applications*. Springer, 2006, pp. 277–286.

[52] W3C, "Web authentication: An api for accessing public key credentials level 1," https://www.w3.org/TR/webauthn/, 2017.

[53] C. Weidenbach, "Towards an automatic analysis of security protocols in first-order logic," in *International Conference on Automated Deduction*. Springer, 1999, pp. 314–328.

[54] Q. Xie, N. Dong, X. Tan, D. S. Wong, and G. Wang, "Improvement of a three-party password-based key exchange protocol with formal verification," *Information Technology and Control*, vol. 42, no. 3, pp. 231–237, 2013.

[55] Y. Zhang, F. Monrose, and M. Reiter, "The security of modern password expiration: An algorithmic framework and empirical analysis," in *ACM conference on Computer and communications security*. ACM, 2010, pp. 176–186.

[56] Y. Zhang, X. Wang, Z. Zhao, and H. Li, "Secure display for fido transaction confirmation," in *ACM Conference on Data and Application Security and Privacy*, 2018, pp. 155–157.

[57] S. Ziauddin and B. Martin, "Formal analysis of iso/iec 9798-2 authentication standard using avispa," in *Asia joint conference on information security*. IEEE, 2013, pp. 108–114.