FIDO Gets Verified: A Formal Analysis of the Universal Authentication Framework Protocol

Haonan Feng*, Jingjing Guan*, Hui Li, Xuesong Pan, and Ziming Zhao, *Member, IEEE*

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 two practical attacks on two 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.

Index Terms—FIDO UAF, formal methods, authentication protocol

1 Introduction

Fast IDentity Online (FIDO) has gained significant popularity in recent years as a public-key cryptography-based authentication framework that enables users to log in to *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 [12], [13], [56]. At the time of writing, more than 250 companies have become members of the FIDO alliance [25], and more than 703 certified FIDO products are in the market [35]. 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 [22].

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) [53] and Client-to-Authenticator Protocol (CTAP) [23]. 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 [15], [37], [43], [44], [48]. 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 [39], [49]. 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 and comprehensively verifying the security properties of the UAF protocol is challenging and time-consuming: 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.

We tackle the aforementioned challenges and resort to formal methods, which have been used in verifying the security of real-world protocols, such as Needham-Schroeder [45], TLS [6]–[8], 5G authentication [4], IKE [16], Diffie-Hellman [1], [50], ISO/IEC 9798-2 [58], LMAP [38],

A preliminary version of this manuscript titled "A Formal Analysis of the FIDO UAF Protocol" was published in the Proceedings of Network and Distributed System Security Symposium (NDSS) 2021.
H. Feng, J. Guan, H. Li, and X. Pan are with the School of Cyberspace Security, Beijing University of Posts and Telecommunications, China. E-mail: {fenghaonan, guaner, lihuill, panxuesong}@bupt.edu.cn.
Z. Zhao is with the Department of Computer Science and Engineering, University at Buffalo, USA. E-mail: zimingzh@buffalo.edu.
**Haonan Feng and Jingjing Guan contributed equally to this manuscript. The corresponding author is Hui Li.*

vTPM migration [14], 3PAKE [55], e-voting [18], E-Health [19], USB Type-C [51], 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 (§3);

2) We provide a *faithful, detailed, and formal model of the UAF protocol*, which is the most comprehensive representation of UAF in literature (§4);

3) We carry out an *automatic verification in the symbolic model* using ProVerif [9], [10]. We open-source our verification code UAFVerif+ and AutoMinAs[1], which are front-end tools to ProVerif. UAFVerif+ supports the analysis of a large number of UAF use cases (§4), and AutoMinAs is a generic tool for identifying minimal assumptions;

4) We present the *verification results* and *minimal assumptions* for UAF to meet each security property (§5);

5) We present *theoretical attacks against each security property* in the form of adversarial capability combinations (§6.1);

6) Guided by the theoretical attacks, we present five practical attacks and case studies. We successfully conducted two of these attacks on two popular Android apps. We responsibly disclosed vulnerabilities to the vendors, and a medium-risk vulnerability ID was assigned (§6.2);

7) We offer *several concrete and explicit suggestions* to fix the identified problems (§7).

This paper is an extended version of our previous work [21] published in the Network and Distributed System Security Symposium (NDSS) 2021. The main differences between these two versions are listed below: i) we refine our formal model to cover more use cases defined in the specifications. We clarify the operations of the web server (WS) and the UAF server (US) inside the Relying Party (RP) and the messages between them, and consider the scenarios that the UAF Server does not provide *AppID* in the request; ii) we present a new method of identifying the minimal assumptions with the correlation assertions in π-calculus, and develop UAFVerif+. Our experimental results show that UAFVerif+ is more efficient than UAFVerif [21]. We also develop AutoMinAs, which can automatically obtain the minimal assumptions for a given protocol taking the protocol process, security assumptions, and security properties as inputs; iii) we present all theoretical attacks corresponding to each security goal, which are presented in the form of atomic adversarial capability combinations. We also discuss our newly discovered attacks in the refined model.

2 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., a 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 passwords. 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 1 describes the acronyms used in this paper. Section 2.1 presents the overall architecture and entities of UAF. Some of the steps and exchanged messages of the protocol differ based on the type of authenticators in use. In Section 2.2 and Section 2.3, we illustrate the protocol using 1st-factor bound authenticators [27]. Section 2.4 explains the protocol operations under different types of authenticators.

2.1 Architecture

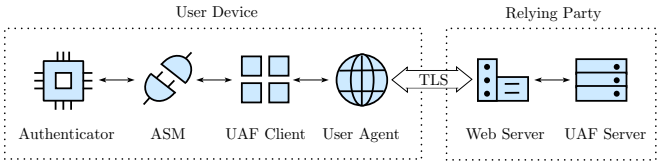

Fig. 1. UAF architecture

As shown in Figure 1, we abstract six major entities and five communication channels in the Universal Authentication Framework. On the user side:

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}) [28]. 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 2.4;

2) the *Authenticator Specific Module* (ASM) is an abstraction layer that provides a uniform API. When ASM is launched for the first time, it generates a secret (tok) [29];

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 retrieves 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;

The Relying Party (RP) consists of a web server and a UAF server: 5) a *Web Server* (WS) interacts with the User Agent and processes login requests from the user with the original login method. Web Server also provides the required information to UAF Server and forwards the UAF protocol messages; 6) a *UAF Server* (US) handles the UAF requests, ensures that only trusted authenticators can be

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

Acronym	Full name	Description
RP	Relying Party	The server-side, which contains a web server and a UAF server.
WS	Web Server	A part of the Relying Party that interacts with the User Agent and UAF Server and forwards the UAF protocol messages.
US	UAF Server	A part of the Relying Party that handles the UAF requests, ensures that only trusted authenticators can be registered, manages the association of authenticators to user accounts, and evaluates user authentications.
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).
CallerID	Caller Identifier	The ID the platform has assigned to the calling UAF Client. On different platforms, the *CallerID* can be obtained differently.
PersonaID	Persona Identifier	An identifier assigned to every operating system user account, which is obtained by the ASM from the operational environment.
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. 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 a private authentication 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 asymmetric public 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 1
Acronyms and descriptions.

registered, manages the association of authenticators to user accounts, and evaluates user authentications.

It is necessary to separate the operations of WS and US, and analyze the messages between them because WS and US are not necessarily in the same trusted domain. WS directly communicates with UA, it can obtain sensitive data, such as *TLSData*, the information from the TLS channel of UA, *UName*, and the inputs from users during the login process with the original authentication method.

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

2.2 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, WS gets the *UName* and TLS channel information *TLSData* of this session. Then WS sends them to US to prevent the TLS MITM attack [40]. US stores *UName* and *TLSData*, generates a registration request $\langle$*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 from US, UC retrieves the trusted user agent list from *AppID* and verifies if *FacetID* is on the list [24]. When US does not provide *AppID*, UC does not verify the *FacetID* but sets the *AppID* to the *FacetID*. UC stores the session ID *SData* as *xSData* and obtains *TSLData* of the TLS channel. Then, UC sends *UName* and the final challenge $fcp = \langle$*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* $||$ *PersonaID* $)$, 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 [29]. 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 verify 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 US. *CNTR* can be used by US to detect cloned authenticators. Finally, the authenticator computes the signature $S = \texttt{sign}_{sk_{AT}}($*AAID*, fc, *KeyID*, $CNTR_A$, $pk_{AU}$$)$, where 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 US.

To verify US is in the same session with UA and UC, US 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* with those stored in US, and checks if *fcp.FacetID* is in the trusted FacetIDs list. Then, US verifies the signature S with the attestation public key (pk_{AT}) of this authenticator. If the signature matches, US stores $CNTR_A$, pk_{AU}, *KeyID*, and *AAID* and completes this registration.

2.3 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, the user initiates authentication with the UA on the device and WS can obtain the *UName* of this session. After receiving the *UName* and *TLSData* from WS, US sends an authentication request $\langle$*AppID*, *KeyID*, *SData*, *Chlg*, [*Tr*]$\rangle$ to UC. Then UC verifies *FacetID*, computes *fcp* $= \langle$*AppID*, *Facet*, *Chlg*, *TLSData*$\rangle$, and sends $\langle$*fcp*, *KeyID*, [*Tr*]$\rangle$ to ASM.

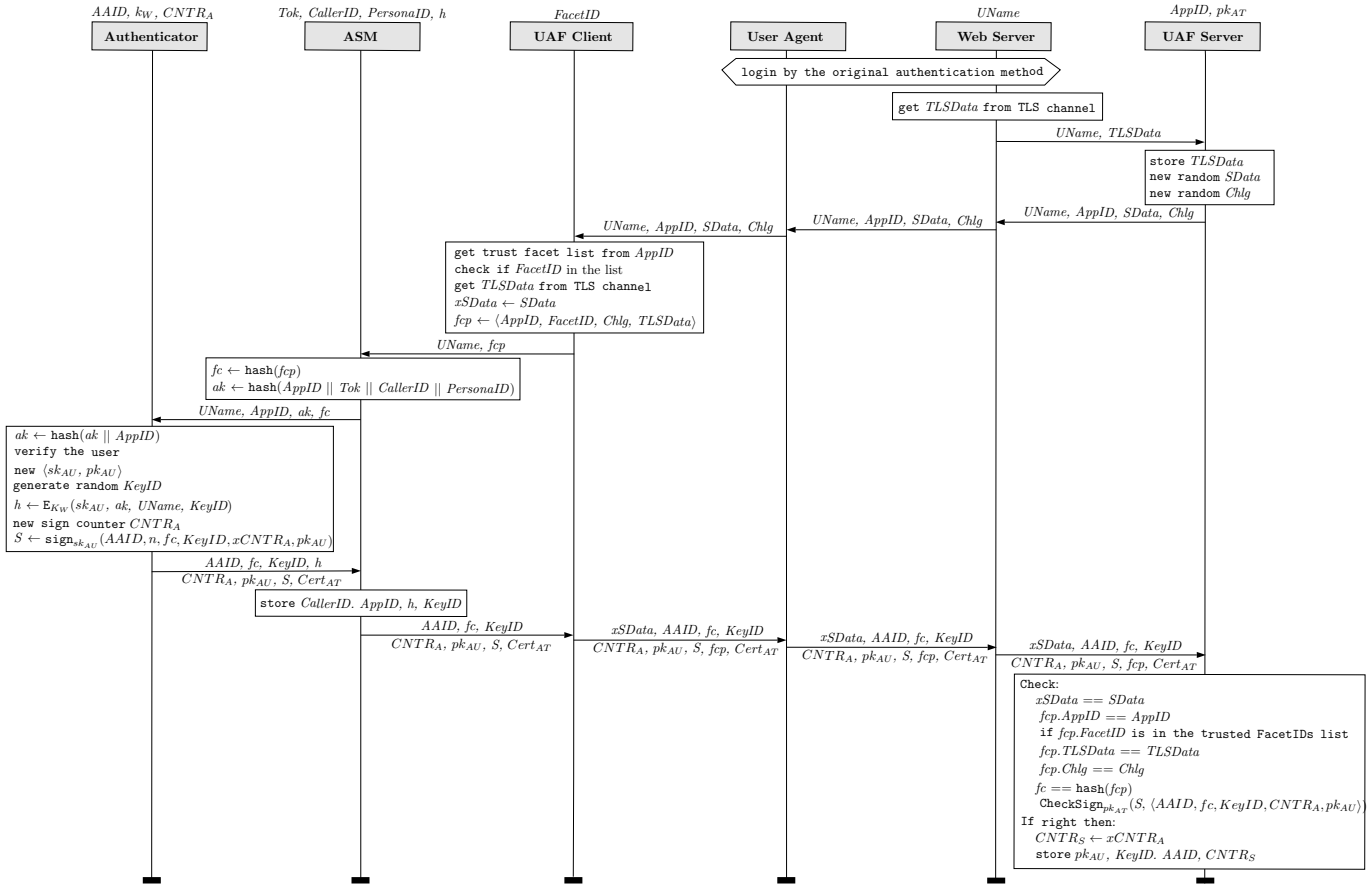

Fig. 2. Registration of the Authenticator

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

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

US locates pk_{AU} of the user by ⟨*UName*, *AAID'*, *KeyID*⟩. It compares *SData* and *xSData* to make sure it is the same session. Next, it verifies *fcp.AppID*, *fcp.Chlg* and *fcp.TLSData* correspond to those stored in US, 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 US. Then US computes $xfc = \texttt{hash}$ (*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, US checks whether $xCNTR_A$ increases compared to $CNTR_S$, if not, the sync fails. If all the checks pass, US updates $CNTR_S$ with $xCNTR_A$ and finishes the authentication process.

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

TABLE 2
Differences under different types of authenticators and use cases.

2.4 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 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. When there is no user session (no cookies), US communicates with UA for the first time and does not know who the user is and cannot provide *KeyID* associated with the user. We call this case login authentication. The other case is called step-up

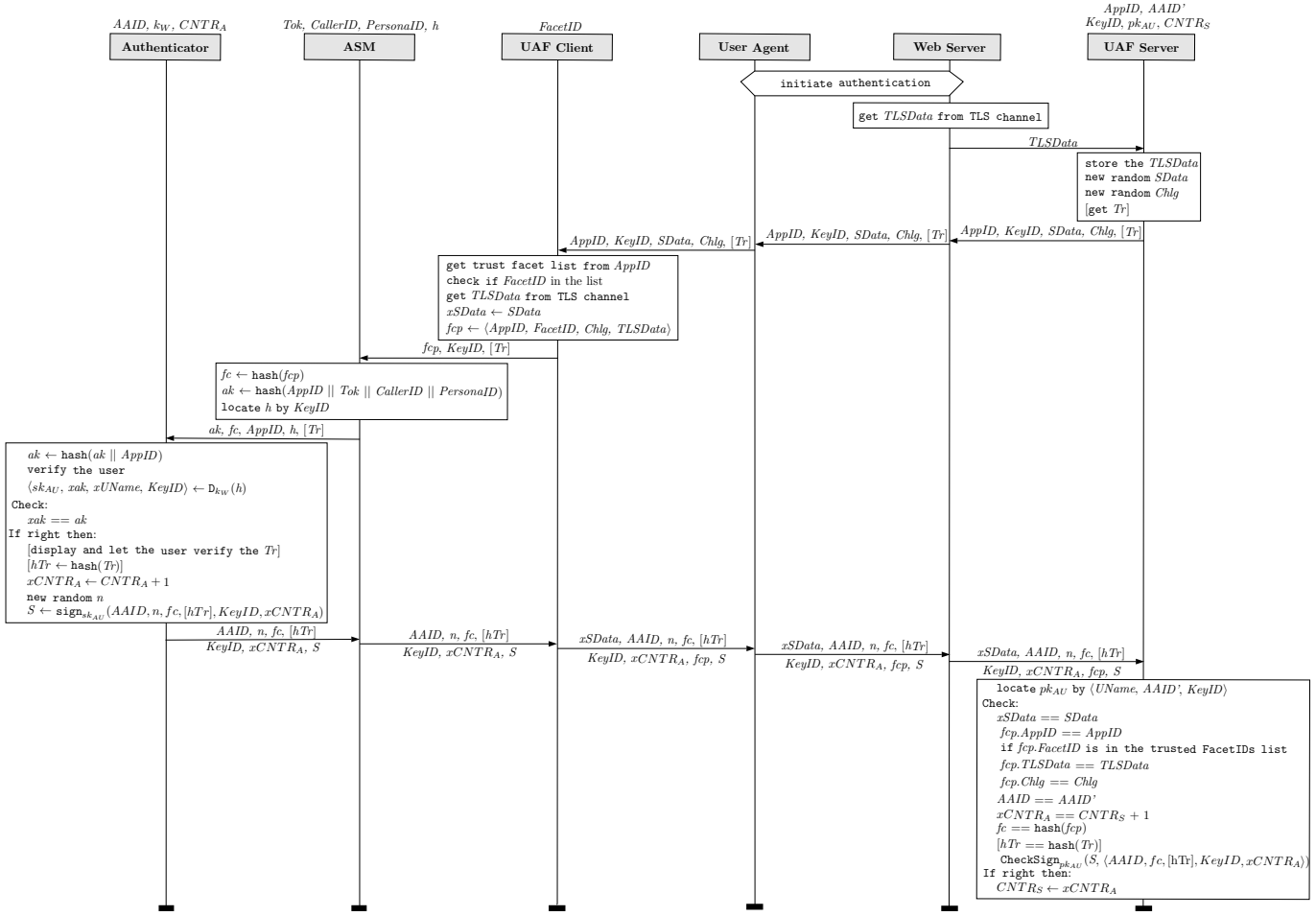

Fig. 3. Authentication operation: the operations framed by '[]' are needed only in the transaction confirmation operations.

authentication, where there is already a user session. US 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 [28]. Table 2 summarizes the differences: i) only 1st-factor authenticators (1B and 1R) can be used in login authentication cases; ii) 2R 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 US; 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, *CallerID*, and *PersonaID*. For roaming authenticators, ak only contains *AppID*; vii) in the authentication process, US provides *KeyID* only for step-up authentication cases.

In addition, US may not provide *AppID* in its requests. When US provides *AppID*, UC retrieves the trusted user agent list from *AppID* and verifies if *FacetID* is on the list. When US does not provide *AppID*, UC sets the *AppID* to *FacetID* during the registration process, US stores this *FacetID*, and the account cannot be used by other application facets. In authentication process, UC still sets the *AppID* to *FacetID*, US will check whether it matches the *FacetID* stored during the registration process.

Considering the cases where the US does not provide *AppID* covers more scenarios defined in the specifications. The necessity of providing *AppID* can be further illustrated by comparing the verification results in the scenarios where *AppID* is provided or not.

3 Threat Model and Security Goals

The UAF specifications list their security assumptions in Section 6 of the security reference [26] and provide the allowed cryptography list [31], the allowed operating environment list [32], the authenticator metadata requirements [33], and the authenticator security requirements [34] in the 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 [26]–[29].

3.1 Assumptions and Threat Model

3.1.1 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.

3.1.2 Assumptions on Channels and Entities

UAF uses five 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 WS use a network channel, which is protected by the TLS. Unlike the DY model, which grants the attacker full control over the network, the attacker in the UAF model should only control the channel between malicious entities. 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, which could communicate with a 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 [41], [42].

3.1.3 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}, and sk_{AU}.

3.1.4 Assumptions on Authenticators

We assume the authenticator always authenticates users correctly, so US authenticating the authenticator is equivalent to US authenticating the user. Because a large number of authenticators may share the same *AAID* and sk_{AT} for privacy preserving [30], 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 US.

3.2 Formalization of UAF's Security Goals

The formal expressions are indicated in *italic text*.

3.2.1 Authentication Properties

To precisely formalize the authentication properties, we resort to Lowe's taxonomy of authentication properties [46], which can be directly modeled in formal methods and widely used in previous research [4]. 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 [26]. The security goals SG-1 to SG-9 do not specify the level of authentication that UAF should provide. With the descriptions of attack resistance in SG-10 to SG-13, we derive that the authentication in UAF should be injective agreement. 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, US (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, *US 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 log in 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 a 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 invalid information will not affect the user's authentication, which means the attacker cannot pass the authentication of US, thus making *CNTR* of US and *CNTR* of the authenticator out of sync. Therefore, the protocol must ensure that every successful authentication of US 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, *US 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, US must obtain injective agreement on UName, AAID, KeyID, AppID with the authenticator when another US 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.*

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, US must obtain injective agreement on UName, AAID with UA when another US 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 the registration request must have been initiated by a legitimate user. We assume the legitimate UA that initiated the registration request can represent the consent of the user, so *US must obtain injective agreement on UName, AppID with UA after the registration process.*

However, even though 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 *US 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 with the transaction message should be detected, and the user cannot deny the transaction message, as in SG-14.

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

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

3.2.2 Confidentiality Properties

The confidentiality of sk_{AT}, sk_{AU}, and k_W is required in Section 4.1 of the security reference [26]. 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 can impersonate ASM and call the authenticator. ASM should maintain the secrecy of ak, as in Section 6.1 [29]. Formally, *ak should remain secret in the presence of the active attacker during the registration and the authentication process.*

3.2.3 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 user behaviors.

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:

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 USs. 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 USes.*

4 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 [47], AVISPA [2], can also be used to model and verify UAF. We choose ProVerif due to its popularity and ease of use.

4.1 Overview of ProVerif

ProVerif is an automatic symbolic protocol verifier, which can verify various security properties, including confidentiality, authentication, and observational equivalence [9]. Comparing with other tools, such as AVISPA [2], CL-AtSe [52], OFMC [5], Tamarin [47], FDR [45], Scyther [17], SATMC [3], Cryptyc [36], TA4SP [11], Maude-NPA [20], 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 be used to verify the UAF protocol since the protocol does not use such operations.

ProVerif verifies in an extension of the applied π-calculus with cryptography. Based on first-order logic resolution rules on Horn clauses [54], it determines whether the properties are met. If a property is violated, it constructs an attack at both the Horn clause level and the π-calculus level. In π-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:

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

4.2 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. In the following example, ProVerif tests the confidentiality of the term `M` and the confidentiality of `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 [9]. 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, where P uses the first component of 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 WS. To model this situation, we need to make sure the attacker knows what WS 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, which WS 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 WS and UA is public, which allows the attacker to have the same knowledge as WS; iii) we assume the attacker is a passive attacker who could only listen to the communication channel between WS and UA.

4.3 ProVerif Models of the UAF

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

4.3.1 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, neither of them can directly model the 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 process `A'`, there is no malicious `B` in the environment.

This model becomes more complex in UAF because it contains many entities. For example, ASM communicates with two entities, UC and 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 an 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 address 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)
    )
)
```

4.3.2 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 RP registers or authenticates a user. By controlling the arguments of the System, we specify which RP 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.

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 are different because the authenticator always generates new sk_{AU} and *KeyID* for each account in registration. Since different users may use authenticators with the same *AAID*, we do not consider the differences in *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* on unlinkability.

```
process
    ...
    !system(AppID,AAID,skAU,KeyID,kW,tok,UName,
        FacetID,CallerID,PersonaID,CNTR)|
    !system(AppID2,AAID,skAU2,KeyID2,choice[kW,kW2],
        choice[tok,tok2],UName2,FacetID2,CallerID,
        PersonaID,CNTR2)
```

4.3.3 Identifying Minimal Assumptions

In our previous work, we obtained the minimal assumptions of the security properties by verifying the cases with different combinations of assumptions. We defined a variable security set $\mathcal{A}$, where $\mathcal{A} = \emptyset$ represents no security assumption. Our tool UAFVerif, which is a front-end of ProVerif, can add security assumptions $\{a_1, ..., a_n\}$ into $\mathcal{A}$ to generate different scenarios and invoke ProVerif to analyze if the protocol satisfies the security properties under these security assumptions. However, a specific copy of ProVerif code is required for each case, and the read and write operations of files increase the overhead of time.

With the help of *correspondence assertions* in π-calculus, we can identify the minimal assumptions with a single file and reduce the overhead caused by frequent file operations. We compare the performance of UAFVerif and UAFVerif+ on the same computer with Intel(R) Core(TM) i7-6500U CPU and 8GB RAM. As shown in Table 3, UAFVerif+ needs more code to model and analyze more scenarios in the refined formal model. While the number of files to be analyzed is greatly reduced, as UAFVerif+ only generates files for each security goal, not each scenario. Meanwhile, it only takes half the time of UAFVerif to get all the analysis results.

Version	Codes (line)	Cases	Files	Time (h:min)
UAFVerif	684 (Python) + 594 (ProVerif)	417,792	417,792	10:57
UAFVerif+	1024 (Python) + 992 (ProVerif)	696,284	104	4:17

TABLE 3
Comparison of UAFVerif and UAFVerif+.

Our method is not limited to the analysis of UAF. We also developed a tool AutoMinAs to automatically identify minimal assumptions for any other protocols. The protocol analysts only need to specify the protocol process, the assumptions, and the security goals of a protocol. AutoMinAs can automatically generate all the correspondence assertions and ProVerif queries for each security goal and then invokes ProVerif to obtain the results, which are further processed to determine the minimal assumptions.

We define *events* for compromised scenarios, including the leakage of data fields and the existence of malicious entities. For example, if the event "leak_skau()" is executed, then the attacker knows sk_{AU}. Similarly, if the event "malicious_UC_to_ASM()" is executed, then the attacker can act as a malicious UC to communicate with the honest ASM.

```
free c:channel[public].
event leak_skau().
event leak_cntr().
event malicious_UC_to_ASM().
let ASM(MC:channel,MA:channel) = (...)
let UC(CM:channel) = (...)
process
!(
    new MA:channel;
    new MC:channel;
    (event leak_skau(); out(c,skAU))|(*leak skAU*)
    (event leak_cntr(); out(c,CNTR))|(*leak CNTR*)
    ASM(MC,MA)|(* honest ASM *)
    (event malicious_UC_to_ASM(); ASM(c,MA))|(*ASM
        communicate with malicious UC*)
    UC(MC)(*honest UAF client*)
)
```

We can add different combinations of assumptions to set $\mathcal{A}$ by appending corresponding combinations of events to the query statement. Each query statement represents a specific scenario. For example, when verifying the confidentiality of *Tr* under the scenario without any assumptions (i.e., all considered data fields are leaked, and all malicious entities exist), we can use the following query statement:

```
query attacker(new Tr).
```

If this query statement is true, then the protocol holds this property under the scenario with no assumption. When we need to analyze the protocol under the scenario that the sk_{AU} is not leaked, we can use the following statement:

```
query attacker(new Tr) ==> event(leak_skau).
```

The query statement verifies such a fact that when the attacker can obtain *Tr*, then sk_{AU} must have been leaked. If this statement is true, the secrecy of sk_{AU} is an assumption for the confidentiality of *Tr*. Then, when analyzing the protocol under the scenario that sk_{AU} and k_W are both compromised, we can use the following statement:

```
query attacker(new Tr) ==> event(leak_skau) || event
    (leak_kw).
```

This statement verifies such a fact that when the attacker can obtain *Tr*, then either sk_{AU} leaks, k_W leaks, or both of them leak. If this statement is true, then the secrecy of sk_{AU} and k_W is an assumption for the confidentiality of *Tr*.

After verifying the query of each property with all combinations of events, we can obtain a series of combinations of assumptions required for the properties. According to these results, we can obtain the minimal assumptions required for each property. More formally, with the combinations of assumptions required for the properties, we can obtain the set A of assumptions that makes the properties holds. When each proper subset of A cannot satisfy a security property, we say satisfying A is a minimal assumption for the protocol to satisfy this security property.

5 Security Analysis

In this section, we present the formal verification results of the UAF protocol. We identify the minimal security assumptions required for each security goal 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 696,284 automatically generated cases considering different authenticator types, scenarios, and assumptions. It took 4 hours to analyze all cases on a computer with Intel(R) Core(TM) i7-6500U CPU and 8GB RAM.

5.1 Result Overview

Reg.	Type	1B	2B	1R	2R
C.	k_W	$\surd$			
	sk_{AT}	$\surd$			
	sk_{AU}	$\neg k_W \vee \neg M[A]$		$\surd$	$\neg k_W \vee (\neg S[W] \wedge \neg C[M] \wedge \neg M[A])$
	ak	$\neg tok \wedge \neg A[M]$		$\times$	
	CNTR	$\neg S[W] \wedge \neg C[M] \wedge \neg M[A]$			
	CNTR (non-a)	$\neg S[W] \wedge \neg U[C] \wedge \neg C[M] \wedge \neg M[A]$			
A.	Basic	$\neg W[S] \wedge \neg C[U] \wedge \neg M[C] \wedge \neg A[M]$			

TABLE 4
Minimal assumptions required for the UAF registration process to achieve confidentiality properties and authentication properties.

Table 4 - 6 presents 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 3.2, 'Non-R' represents the transaction non-repudiation goal. 'non-a' represents the goals under the protocol that US does not provide *AppID*.

We present security assumptions in symbols: '$\wedge$' denotes AND, whereas '$\vee$' denotes OR. *A*, *M*, *C*, *U*, *W*, and *S* represent Authenticator, ASM, UC, UA, WS, and US respectively. '$\neg$' before a data field represents the field is not compromised. '$\neg$' before '$X[Y]$' represents that the system does not have 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. '$\times$' means the protocol cannot meet the security goal. '$-$' means we do not consider this property.

5.2 Confidentiality Properties

As Table 4 shows, the protocol does not disclose k_W or sk_{AT} in registration 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 in its internal storage). 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 US. Therefore, the protocol should guarantee that k_W will not be compromised, or the protocol satisfies $\neg S[W]$ and $\neg C[M]$ and $\neg M[A]$ to ensure the h does not leak. As Table 5 shows in authentication, 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

Auth.	Type	1B		2B
		login	step-up	step-up
C.	sk_{AU}	$\neg k_W \vee \neg A[M]$		
	ak	$\neg tok \wedge \neg A[M]$		
	CNTR	$(\neg S[W] \wedge \neg W[U] \wedge \neg M[A]) \vee (\neg S[W] \wedge \neg W[U] \wedge \neg A[M] \wedge \neg k_W)$		$(\neg S[W] \wedge \neg M[A]) \vee (\neg S[W] \wedge \neg A[M] \wedge \neg k_W)$
	CNTR(non-a)	$(\neg S[W] \wedge W[U] \wedge \neg U[C] \wedge \neg M[A]) \vee (\neg S[W] \wedge W[U] \wedge \neg U[C] \wedge \neg A[M] \wedge \neg k_W)$		$(\neg S[W] \wedge \neg U[C] \wedge \neg M[A]) \vee (\neg S[W] \wedge \neg U[C] \wedge \neg A[M] \wedge \neg k_W)$
	Tr	$\neg W[S] \wedge C[U] \wedge \neg M[C] \wedge \neg A[M]$		
A.	Basic	$(\neg sk_{AU} \wedge \neg A[M]) \vee (\neg sk_{AU} \wedge \neg k_W \wedge \neg M[A])$		$(\neg sk_{AU} \wedge \neg A[M]) \vee (\neg sk_{AU} \wedge \neg k_W \wedge \neg M[A]) \vee (\neg sk_{AU} \wedge \neg k_W \wedge \neg W[S]) \vee (\neg W[S] \wedge \neg C[U] \wedge \neg M[C] \wedge \neg A[M])$
	Non-R	-		$(\neg sk_{AU} \wedge \neg A[M]) \vee (\neg sk_{AU} \wedge \neg k_W \wedge \neg M[A]) \vee (\neg sk_{AU} \wedge \neg k_W \wedge \neg W[S]) \vee (\neg W[S] \wedge \neg C[U] \wedge \neg M[C] \wedge \neg A[M])$

TABLE 5
Minimal assumptions required for the UAF authentication process to achieve confidentiality properties and authentication properties.

Auth.	Type	1R		2R
		login	step-up	step-up
C.	sk_{AU}	$\surd$		$\neg k_W$
	ak	$\times$		
	CNTR	$(\neg S[W] \wedge \neg W[U] \wedge \neg C[M] \wedge \neg M[A])$	$\neg S[W] \wedge \neg C[M] \wedge \neg M[A]$	
	CNTR (non-a)	$\neg S[W] \wedge \neg W[U] \wedge \neg U[C] \wedge \neg C[M] \wedge \neg M[A])$	$\neg S[W] \wedge \neg U[C] \wedge \neg C[M] \wedge \neg M[A]$	
	Tr	$\neg W[S] \wedge C[U] \wedge \neg M[C] \wedge \neg A[M]$		
A.	Basic	$(\neg sk_{AU} \wedge \neg C[M] \wedge \neg M[A])$	$(\neg sk_{AU} \wedge \neg W[S]) \vee (\neg sk_{AU} \wedge \neg C[M] \wedge \neg M[A]) \vee (\neg W[S] \wedge \neg C[U] \wedge \neg M[C] \wedge \neg A[M])$	$(\neg sk_{AU} \wedge \neg k_W \wedge \neg W[S]) \vee (\neg sk_{AU} \wedge \neg k_W \wedge \neg C[M] \wedge \neg M[A]) \vee (\neg W[S] \wedge \neg C[U] \wedge \neg M[C] \wedge \neg A[M])$
	Non-R	-	$(\neg sk_{AU} \wedge \neg W[S]) \vee (\neg sk_{AU} \wedge \neg C[M] \wedge \neg M[A]) \vee (\neg W[S] \wedge \neg C[U] \wedge \neg M[C] \wedge \neg A[M])$	$(\neg sk_{AU} \wedge \neg k_W \wedge \neg W[S]) \vee (\neg sk_{AU} \wedge \neg k_W \wedge \neg C[M] \wedge \neg M[A]) \vee (\neg W[S] \wedge \neg C[U] \wedge \neg M[C] \wedge \neg A[M])$

TABLE 6
Minimal assumptions required for the UAF authentication process to achieve confidentiality properties and authentication properties2.

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 US, 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 $ak = \texttt{hash}(\textit{AppID} \parallel tok \parallel \textit{CallerID} \parallel \textit{PersonaID})$. 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, the confidentiality of ak 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 6 and provide a fix in Section 7.2.

To maintain the confidentiality of *CNTR* in registration, the deployment of the protocol must satisfy $\neg S[W]$ and $\neg C[M]$ and $\neg M[A]$ when US provides *AppID*. Otherwise, a malicious entity can initiate a registration process to get *CNTR* from the response generated by the authenticator. When US does not provide *AppID*, the minimal assumption additionally needs $\neg U[C]$ since the UC cannot verify UA.

To maintain the confidentiality of *CNTR* in the authentication process when US provides *AppID*, for 1B authenticators when login, the protocol needs to satisfy $\neg S[W]$ and $\neg W[U]$ and $\neg M[A]$, or satisfy $\neg S[W]$ and $\neg W[U]$ and $\neg A[M]$ and k_W. Otherwise, the malicious entity can initiate an authentication process to get *CNTR* from the response generated by a malicious authenticator. Since ASM has access control over UC's *CallerID* and UC verifies UA by *AppID*, malicious UCs and UAs cannot start the legitimate authenticator and get *CNTR*. Without knowing k_W, the attacker cannot calculate h, and the attacker cannot obtain h if the protocol holds $\neg A[M]$, even there is a malicious ASM, the attacker cannot start the authenticator. So, the assumptions $\neg M[A]$ can be replaced by $\neg A[M]$ and $\neg k_W$. The assumptions when step-up authentication in 1B and 2B authenticators are similar, except that there is no $W[U]$ since we assume that the step-up authentication is based on a trusted connection between WS and UA (already satisfies $\neg U[W]$ and $\neg W[U]$). For 1R authenticator when login, since ASM does not verify UC, so the protocol additionally requires $\neg C[M]$. Besides, h is stored in the authenticator rather than ASM, and ak only contains *AppID* which is public to the attacker, so as long as there is a malicious ASM, the attacker can start the authenticator and obtain *CNTR*. For 1R and 2R authenticators when step-up authentication, there is no $W[U]$ since we assume that the step-up authentication is based on a trusted connection between WS and UA. To maintain the confidentiality of *CNTR* in the authentication process when US does not provide *AppID*, the protocol additionally requires $\neg U[C]$ since the UC cannot verify UA.

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

5.3 Authentication Properties

As shown in Table 4, to achieve the authentication goals in registration, the minimal assumption is $\neg W[S]$ and $\neg C[U]$ 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 US 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 $\neg W[S]$ and

$\neg C[U]$ and $\neg M[C]$ and $\neg A[M]$. ProVerif generates an attack when one of them is not satisfied, which we will discuss in Section 6. The root cause of this issue is that there is no access control mechanism from WS to US, from US to WS, 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.

As shown in Table 5 and 6, during authentication, the minimal assumptions vary under different types of authenticators and use cases. When using 1B and 2B authenticators, we found that holding $\neg sk_{AU}$ and $\neg A[M]$ is an intuitive way to meet the authentication goal. That is because if there is no malicious authenticator, the attacker cannot obtain h from ASM and cannot decrypt it to get sk_{AU}. And if the attacker does not know sk_{AU}, he/she cannot construct the signature and impersonate the user. Note that the assumption $\neg A[M]$ can be replaced by $\neg k_W$ and $\neg M[A]$ for the same reason as above. For 1B and 2B authenticators when step-up authentication, since the protocol is based on successful login, the attacker cannot use a malicious UA to send messages to WS. As long as the attack cannot obtain sk_{AU} and there is no malicious WS that can communicate with the honest US, the attacker cannot break the authentication goal. In addition, even if sk_{AU} is leaked, if the protocol meets $\neg W[S]$ and $\neg C[U]$ and $\neg M[C]$ and $\neg A[M]$, the attacker cannot send messages to the US and finish the authentication, and the authentication goal holds.

1R authenticator stores h itself, and ak is only the hash of *AppID* known by the attacker, so as long as the attacker can send the request to the authenticator, he/she can get the signature and implement the parallel session attack. Compared with 1B authenticators, for 1R authenticators when login, $\neg C[M]$ is additionally needed, or the attacker can use a malicious UC to send the request to the authenticator.

For 1R authenticator in step-up authentication, when the protocol satisfies $\neg sk_{AU}$ and $\neg C[M]$ and $\neg M[A]$, the authentication goal holds. Besides, since the step-up authentication is based on a trust connection (already satisfies $\neg U[W]$ and $\neg W[U]$), when the protocol satisfies $\neg sk_{AU}$ and $\neg W[S]$, even if a malicious entity can act as a middleman, the attacker inevitably needs to start the honest authenticator and forwards the message to the honest UA to send to US, which does not break the authentication goals. When the protocol satisfies $\neg W[S]$ and $\neg C[U]$ and $\neg M[C]$ and $\neg A[M]$, the attacker cannot intercept a message and send the response, and the authentication goals hold.

For 2R authenticators in step-up authentication, when the protocol satisfies $\neg W[S]$ and $\neg C[U]$ and $\neg M[C]$ and $\neg A[M]$, the attacker cannot intercept messages. Compared with 1R authenticators, for 2R authenticators, h is regarded as $KeyID$, kept in US, and is sent to the authenticator during the authentication process. Once the protocol does not satisfy $\neg k_W$, the attacker can obtain h by a malicious entity and obtain sk_{AU} by decrypting h to break the authentication goals. Therefore, the protocol needs to satisfy $\neg sk_{AU}$ and k_W and $\neg W[S]$, or $\neg sk_{AU}$ and k_W and $\neg C[M]$ and $\neg M[A]$.

The minimal assumptions to achieve the non-repudiation goal are the same as the authentication goals. Whether *AppID* is provided or not does not affect the minimal assumptions of the authentication goals, because US still verifies *FacetID* when *AppID* is not provided.

The results show that when the attacker cannot compromise US, and there are no malicious USs, the protocol can prevent attacks from malicious WS (that can communicate with UA) and malicious UA (that can communicate with UC) 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 WS. In this situation, *FacetID*, which is the identifier of the malicious UA, is not in the trusted user agent list retrieved from *AppID*. However, when the honest RP uses the third-party US rather than deploying US itself, WS may communicate with the malicious US and break the authentication goals.

From the results, we found that the attestation mechanism (signed by sk_{AT}) can only ensure that it is a legitimate authenticator who completes the registration but cannot guarantee that it must be the user's authenticator, i.e., anyone with a legitimate authenticator can finish registration (including the attacker's authenticator), which may lead to the attacker binding user account on his/her authenticator. Note that there are no other authentication measures for the registration process. So the registration process is more vulnerable to attacks than the authentication process.

5.4 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*, and *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.

6 Attacks on the UAF Protocol

The minimal assumptions in Section 5, which are represented by the combinations of limitations on atomic adversarial capabilities, describe the security of the protocol in terms of the conditions required for satisfying each security property. In this section, we first discuss all theoretical attacks followed by the discussions on five practical attacks.

6.1 Theoretical Attacks

By reversing the expressions in Table 4 - 6, we can obtain the combination of adversarial capabilities to break the security goals and implement attacks against the UAF protocol. We summarize them in Table 7 - 9. 'k_W' means the k_W is compromised. '$X[Y]$' represents that there exists malicious X that can communicate with Y. '$\surd$' means the attacker can break the security goals under any conditions. '$\times$' means there is no way to break the security goals. '$-$' means we do not consider this property in this scenario.

Reg.	Type	1B	2B	1R	2R
C.	k_W	$\times$			
	sk_{AT}	$\times$			
	sk_{AU}	$k_W \wedge M[A]$		$\times$	$(k_W \wedge S[W]) \vee (k_W \wedge M[A]) \vee (k_W \wedge C[M])$
	ak	$tok \vee A[M]$		$\surd$	
	CNTR	$S[W] \vee C[M] \vee M[A]$			
	CNTR (non-a)	$S[W] \vee U[C] \vee C[M] \vee M[A]$			
A.	Basic	$W[S] \vee C[U] \vee M[C] \vee A[M]$			

TABLE 7
Adversarial capabilities to break confidentiality and authentication properties in the UAF registration process.

6.1.1 Break Confidentiality in Registration Process

Rows 2-7 in Table 7 show the adversarial capabilities required to break confidentiality properties in registration.

First, the attacker cannot break the confidentiality of k_W and sk_{AT} unless he/she compromises the authenticator.

For 1B and 2B authenticators, with the capabilities k_W and $M[A]$, the attacker can break the confidentiality of sk_{AU}. The attacker sends the request to the honest authenticator with a malicious ASM and obtains the response containing h, which can be decrypted by k_W to get sk_{AU}. For 1R authenticators, the attacker cannot break the confidentiality of sk_{AU} unless he/she compromises the authenticator. For 2R authenticators, h is assigned to $KeyID$ and sent to US as part of the response. Therefore, as long as the attacker has the capabilities $S[W]$, $C[M]$, or $M[A]$, h will be intercepted by these malicious entities. If the attacker can obtain k_W at the same time, h can be decrypted to obtain sk_{AU}.

For 1B and 2B authenticators, if tok is leaked, the attacker can compute $ak = \texttt{hash}(\textit{AppID} \,||\, tok \,||\, \textit{CallerID} \,||\, \textit{PersonaID})$ and obtain ak. In addition, with the capability $A[M]$, the attacker can use a malicious authenticator to receive the request from the honest ASM, which contains ak. For 1R and 2R authenticators, the attacker can compute $ak = \texttt{hash}(\textit{AppID})$ and obtain ak.

For any type of the authenticators with US providing *AppID*, with the adversarial capabilities $S[W]$, $C[M]$, or $M[A]$, the attacker can obtain *CNTR* by receiving the response with these malicious entities. When US does not provide *AppID*, since UC cannot verify UA, the attacker with capability $U[C]$ can obtain *CNTR* by a malicious UA receiving the response from the honest UC.

6.1.2 Break Authentication in Registration Process

Row 8 in Table 7 shows that, with the capabilities $W[S]$, $C[U]$, $M[C]$, or $A[M]$, the attacker can break the authentication goals in authenticator registration, thereby binding the victim's account with a malicious authenticator. The resulting authenticator rebinding attack will be described in detail in Section 6.2.1.

Auth.	Type	1B		2B
		login	step-up	step-up
C.	sk_{AU}	$k_W \wedge A[M]$		
	ak	$tok \vee A[M]$		
	CNTR	$S[W] \vee W[U] \vee (k_W \wedge M[A]) \vee (A[M] \wedge M[A])$		$S[W] \vee (k_W \wedge M[A]) \vee (A[M] \wedge M[A])$
	CNTR(non-a)	$S[W] \vee W[U] \vee U[C] \vee (k_W \wedge M[A]) \vee (A[M] \wedge M[A])$		$S[W] \vee U[C] \vee (k_W \wedge M[A]) \vee (A[M] \wedge M[A])$
	Tr	$W[S] \vee C[U] \vee M[C] \vee A[M]$		
A.	Basic	$sk_{AU} \vee (A[M] \wedge k_W) \vee (A[M] \wedge M[A])$		$(sk_{AU} \wedge W[S]) \vee (sk_{AU} \wedge C[U]) \vee (sk_{AU} \wedge M[C]) \vee (sk_{AU} \wedge A[M]) \vee (k_W \wedge A[M]) \vee (A[M] \wedge M[A] \wedge W[S])$
	Non-R	-		$(sk_{AU} \wedge W[S]) \vee (sk_{AU} \wedge C[U]) \vee (sk_{AU} \wedge M[C]) \vee (sk_{AU} \wedge A[M]) \vee (k_W \wedge A[M]) \vee (A[M] \wedge M[A] \wedge W[S])$

TABLE 8
Adversarial capabilities to break confidentiality and authentication properties in the UAF authentication process.

Auth.	Type	1R		2R
		login	step-up	step-up
C.	sk_{AU}	$\times$		k_W
	ak	$\surd$		
	CNTR	$S[W] \vee W[U] \vee C[M] \vee M[A]$	$S[W] \vee C[M] \vee M[A]$	
	CNTR (non-a)	$S[W] \vee W[U] \vee U[C] \vee C[M] \vee M[A]$	$S[W] \vee U[C] \vee C[M] \vee M[A]$	
	Tr	$W[S] \vee C[U] \vee M[C] \vee A[M]$		
A.	Basic	$sk_{AU} \vee C[M] \vee M[A]$	$(sk_{AU} \wedge W[S]) \vee (sk_{AU} \wedge C[U]) \vee (sk_{AU} \wedge M[C]) \vee (sk_{AU} \wedge A[M]) \vee (W[S] \wedge C[M]) \vee (W[S] \wedge M[A])$	$(sk_{AU} \wedge W[S]) \vee (sk_{AU} \wedge C[U]) \vee (sk_{AU} \wedge M[C]) \vee (sk_{AU} \wedge A[M]) \vee (k_W \wedge W[S]) \vee (k_W \wedge C[U]) \vee (k_W \wedge M[C]) \vee (k_W \wedge A[M]) \vee (W[S] \wedge C[M]) \vee (W[S] \wedge M[A])$
	Non-R	-	$(sk_{AU} \wedge W[S]) \vee (sk_{AU} \wedge C[U]) \vee (sk_{AU} \wedge M[C]) \vee (sk_{AU} \wedge A[M]) \vee (W[S] \wedge C[M]) \vee (W[S] \wedge M[A])$	$(sk_{AU} \wedge W[S]) \vee (sk_{AU} \wedge C[U]) \vee (sk_{AU} \wedge M[C]) \vee (sk_{AU} \wedge A[M]) \vee (k_W \wedge W[S]) \vee (k_W \wedge C[U]) \vee (k_W \wedge M[C]) \vee (k_W \wedge A[M]) \vee (W[S] \wedge C[M]) \vee (W[S] \wedge M[A])$

TABLE 9
Adversarial capabilities to break confidentiality and authentication properties in the UAF authentication process.

6.1.3 Break Confidentiality in Authentication Process

Rows 2-6 in Table 8 show the adversarial capabilities required to break confidentiality in the UAF authentication process. The attacker can break the confidentiality of sk_{AU} and ak in the same ways as in the registration process.

For 1B authenticators when login and US providing *AppID*, the attacker can break the confidentiality of $CNTR$ with the combinations of capabilities $S[W]$, $W[U]$, k_W and $M[A]$, or $A[M]$ and $M[A]$. The attacker can intercept *CNTR* in the authentication responses with a malicious US or WS. Besides, the attacker can use k_W to construct a fake h' containing a fake ak', and send the request with h' and ak' to the honest authenticator with a malicious ASM. The authenticator extracts the ak' from h' for verification and is unaware of the malicious ASM. The response with *CNTR* will be returned to the malicious ASM. Moreover, with the capabilities $A[M]$ and $M[A]$, the attacker can intercept h and ak with a malicious authenticator, and then use a malicious ASM to send the request and obtain *CNTR*.

Compared with 1B authenticators, for 1R authenticator when login and US providing *AppID*, the attacker can directly use a malicious ASM to obtain *CNTR* since h is stored inside the authenticator and ASM is not required to provide valid h. Besides, since ASM does not verify UC's *CallerID*,

a malicious UC can obtain *CNTR*. Since we assume the WS and UA have already established a trusted channel in step-up authentication, the attackers do not have the capability $W[U]$ in these scenarios. In the scenario where the US does not provide the *AppID*, the attacker with the capability $U[C]$ can also obtain CNTR through a malicious UA.

For any type of authenticators, the attacker with capabilities $W[S]$, $C[U]$, $M[C]$, or $A[M]$ can obtain *Tr* by receiving the request containing *Tr* from honest entities. For example, with the adversarial capability $C[U]$, the attacker can communicate with the honest UA through a malicious UC and intercept *Tr* in the request.

6.1.4 Break Authentication in Authentication Process

For 1B authenticators when login, once sk_{AU} is leaked, the attacker can impersonate a user by forging the signature. With adversarial capabilities k_W and $A[M]$, the attacker can intercept h via the malicious authenticator and decrypt h to obtain the sk_{AU}. As for the cases with adversarial capabilities $A[M]$ and $M[A]$, the attacker can obtain valid h and ak with a malicious authenticator and forge an authentication request to start the honest authenticator. The resulting parallel session attack will be described in Section 6.2.2.

For 1B and 2B authenticators when step-up authentication, since a trusted channel has already been established between UA and WS, the attacker cannot directly communicate with WS using a malicious UA, the attacker cannot break the authentication goals by only compromising the sk_{AU}. However, with the capabilities $W[S]$, $C[U]$, $M[C]$, or $A[M]$ the attacker can forward valid responses via these malicious entities. With the capabilities k_W and $A[M]$, the attacker can obtain h via the malicious authenticator and obtain sk_{AU} by decrypting h with k_W, the forged responses can also be forwarded by the malicious authenticator. The attacker can break the authentication goal or non-repudiation goal with the capabilities $A[M]$, $M[A]$, and $W[S]$, and the resulting transaction tampering attack will be detailed in Section 6.2.5.

For 1R authenticators when login, with the capabilities $C[M]$ or $M[A]$, the attacker can implement parallel session attacks, which we discuss in Section 6.2.2. With the capabilities k_W and $A[M]$, the attacker receives h via the malicious authenticator and obtains sk_{AU} by decrypting h with k_W.

For 1R authenticators when step-up authentication, if sk_{AU} is leaked, the attacker can forge a valid signature. With the adversarial capabilities $W[S]$, $C[U]$, $M[C]$ or $A[M]$, the attacker can send forged responses to break the authentication properties. Similar to 1B and 2B authenticators when step-up authentication, with the capabilities $W[S]$ and $C[M]$, the attacker can use malicious WS to initiate a step-up authentication with US, and use a malicious UC to send requests and obtain the valid signature to bypass the authentication. This is a type of transaction tampering attacks, which will be detailed in Section 6.2.5. With capabilities $W[S]$ and $M[A]$, the attacker can perform a similar attack, except that a malicious ASM is used to send the request.

For 2R authenticators when step-up authentication, which is different from using the 1R authenticator, the $KeyID$ will be assigned to h and forwarded between entities. Therefore, based on the combinations of adversarial capabilities in 1R authenticator scenarios, with the capabilities $W[S]$, $C[U]$, $M[C]$, or $A[M]$ the attacker can also use k_W to decrypt the $KeyID$ and obtain sk_{AU}.

6.2 Practical Attacks and Case Studies

6.2.1 Authenticator Rebinding Attack

When the deployment of the protocol does not meet the assumptions $\neg W[S]$, $\neg C[U]$, $\neg M[C]$, and $\neg A[M]$, or the attacker has the capabilities $W[S]$, $C[U]$, $M[C]$, or $A[M]$, the injective agreement on *UName*, *AAID*, *KeyID*, *AppID* between US 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 US; 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 US and impersonate the victim.

To verify the feasibility of this attack on real-world apps, we compiled a dataset of 1,856 Android payment apps 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 had 214,424,508 downloads in total, and Jingdong Finance had 1,043,164,317 downloads in total. In March 2020, China Mobile Pay had monthly active users of 3,838,000, and Jingdong Finance had monthly active users of 23,116,000.

We reported the vulnerability of China Mobile Pay to 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 they would ignore the vulnerability on December 19, 2018.

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 [37] is similar to the authenticator rebinding attack, except that the former requires the attacker to corrupt UC and ASM. Cloned authenticator attack [48] needs the attacker to evade the security mechanism of the environment of the authenticator, get information about 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).

6.2.2 Parallel Session Attack

Parallel session attacks can be performed during the authentication process when the deployment of the protocol does not satisfy the assumptions $\neg A[M]$ and $\neg M[A]$ (the attacker has the adversarial capabilities $A[M]$ or $M[A]$) for the 1B login case or does not satisfy the assumptions $\neg C[M]$ or $\neg M[A]$ (the attacker has the adversarial capabilities $C[M]$ and $M[A]$) 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 [37], which requires the compromise of the legitimate UC and ASM.

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 $ak =$ `hash(`*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. Similarly, the attacker can impersonate the user and finish the step-up authentication, such as a transaction confirmation.

6.2.3 Privacy Disclosure Attack

When the deployment of the protocol does not satisfy the assumptions $\neg C[U]$ or $\neg C[M]$ (with the adversarial capabilities $C[U]$ and $C[M]$), 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 the 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 use the signature for chosen-ciphertext attacks.

As for the cases where US does not provide *AppID*, the attacker can intercept the *CNTR* with a malicious UA. Since the *AppID* is not used in registration or authentication requests, the UC can not obtain the list of trusted *FacetID*s with *AppID*, and thus can not verify whether the UA in this session is certified by the US. If a malicious UA controlled by the attacker participates in this session, the *CNTR* returned by the authenticator in the responses will be leaked.

6.2.4 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 $\neg A[M]$ (with the adversarial capabilities $C[U]$, $M[C]$, and $A[M]$). 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 US: 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 the message, so *CNTR* does not increment; v) the attacker repeats this attack many times to cause *CNTR* out of sync.

6.2.5 Transaction Tampering Attack

A corrupted WS can carry out the transaction tampering attack when the deployment of the protocol does not satisfy the assumptions $\neg W[S]$, $\neg M[A]$, and $\neg A[M]$ (with the adversarial capabilities $W[S]$, $M[A]$, or $A[M]$) for 1B and 2B authenticators in step-up authentication. The attacker can impersonate the user to finish a transaction confirmation with any Tr without the user knowing.

The transaction tampering attack can be carried out in two steps. In step 1, i) the victim logs in to RP; ii) the victim initiates a step-up authentication with RP; iii) the legitimate ASM forwards the request, which contains ak and h, to the malicious authenticator. In step 2: i) the victim initiates a step-up authentication with RP (e.g., transaction confirmation); ii) the malicious WS initiates a transaction

confirmation with the honest US with any Tr'; iii) the malicious ASM sends the request with Tr', h, and ak to the trusted authenticator, which verifies the user and signs the message; iv) US successfully authenticates the user and finishes the transaction confirmation with Tr'.

7 Recommendations

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

7.1 Explicit Requirements

The security goals in the UAF security reference are informal and fragmentary: i) SG-1 is ambiguous in that 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, 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 (discussed in Section 3.2).

We recommend presenting security requirements and goals more explicitly and actively. For example, the specifications can use formal expressions to describe the security properties as shown in Section 3.2. The analysis results show that the registration process has many issues, so the specifications should improve the description of authentication properties for the registration process.

7.2 Modifying 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 in ASM is based on the Trust On First Use (TOFU) concept [30], 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 from malicious authenticators. We recommend: 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.

7.3 Adding Authentication Mechanism between UAF Entities

UAF protocol has multiple participating entities and lacks inter-entity authentication. The attacker can use malicious entities to participate in the protocol and break the security properties. Therefore, the UAF protocol should add authentication mechanisms between UAF entities. First, WS should authenticate US. When RP does not deploy US itself, an honest WS may communicate with the malicious US. But the UAF standard does not provide any authentication mechanism between WS and US. Once the third-party US is malicious, the attacker can control any user's account. Then, ASM should authenticate UC. 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* [29], 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 having 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. Last, UA should authenticate UC. 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 require UA to authenticate UC. For example, UA can use the same mechanism as ASM does to authenticate UC.

8 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 [37]. Leoutsarakos manually found 15 defects of the UAF protocol, which were not formally verified [43]. Panos et al. presented a manual and informal analysis of the UAF protocol with several discovered vulnerabilities and attacks [48]. 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, a local device computing platform, and the end-user [44]. Zhang et al. presented an attack on FIDO transaction confirmation and proposed a secure display mechanism for mobile devices [57]. 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 [39]. 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 [15]. Pereira et al. formally analyzed the authentication properties of the U2F protocol [49]. 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.

9 Conclusion

In this paper, we formalized the security assumptions and goals of the UAF 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 some attacks. We confirmed previously discovered vulnerabilities in an automated way and disclosed several new attacks. We offered several concrete recommendations to fix the identified problems and weaknesses in UAF.

Acknowledgment

The research of Beijing University and Posts and Telecommunications is partially funded by the Joint funds for Regional Innovation and Development of the National Natural Science Foundation of China (No. U21A20449) and the National Natural Science Foundation of China (Grant No. 61941105).

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.

[2] 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.

[3] 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.

[4] 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.

[5] D. Basin, S. Mödersheim, and L. Vigano, "Ofmc: A symbolic model checker for security protocols," *International Journal of Information Security*, vol. 4, no. 3, 2005.

[6] 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.

[7] 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.

[8] K. Bhargavan, C. Fournet, R. Corin, and E. Zalinescu, "Cryptographically verified implementations for tls," in *ACM conference on Computer and communications security*. ACM, 2008.

[9] B. Blanchet, "Modeling and verifying security protocols with the applied pi calculus and proverif," vol. 1, no. 1-2, 2016.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] C. Cremers, "Key exchange in ipsec revisited: Formal analysis of ikev1 and ikev2," in *European Symposium on Research in Computer Security*. Springer, 2011.

[17] C. J. Cremers, "Unbounded verification, falsification, and characterization of security protocols by pattern refinement," in *ACM conference on Computer and communications security*. ACM, 2008.

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

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

[20] 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, 2006.

[21] H. Feng, H. Li, X. Pan, and Z. Zhao, "A formal analysis of the fido uaf protocol," in *Network And Distributed System Security Symposium*, 2021.

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

[23] FIDO Alliance, "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.

[24] FIDO Alliance, "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.

[25] FIDO Alliance, "FIDO Members," https://fidoalliance.org/members/, 2017.

[26] FIDO Alliance, "FIDO Security Reference," https://fidoalliance.org/specs/fido-uaf-v1.1-ps-20170202/fido-security-ref-v1.1-ps-20170202.html, 2017.

[27] FIDO Alliance, "FIDO UAF Architectural Overview," https://fidoalliance.org/specs/fido-uaf-v1.1-ps-20170202/fido-uaf-overview-v1.1-ps-20170202.html, 2017.

[28] FIDO Alliance, "FIDO UAF Authenticator Commands," https://fidoalliance.org/specs/fido-uaf-v1.1-ps-20170202/fido-uaf-authnr-cmds-v1.1-ps-20170202.html, 2017.

[29] FIDO Alliance, "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.

[30] FIDO Alliance, "FIDO UAF Protocol Specification," https://fidoalliance.org/specs/fido-uaf-v1.1-ps-20170202/fido-uaf-protocol-v1.1-ps-20170202.html, 2017.

[31] FIDO Alliance, "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.

[32] FIDO Alliance, "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.

[33] FIDO Alliance, "FIDO Authenticator Metadata Requirements," https://fidoalliance.org/specs/fido-security-requirements-v1.2-2018/fido-authenticator-metadata-requirements-v1.0-wd-20180629.html, 2018.

[34] FIDO Alliance, "FIDO Authenticator Security Requirements," https://fidoalliance.org/specs/fido-security-requirements/fido-authenticator-security-requirements-v1.3-fd-20180905.html, 2018.

[35] FIDO Alliance, "FIDO Certified Products," https://fidoalliance.org/certification/fido-certified-products/, 2019.

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

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

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

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

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

[41] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual international cryptology conference*. Springer, 1999.

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

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

[44] I. Loutfi and A. Jøsang, "Fido trust requirements," in *Nordic Conference on Secure IT Systems*. Springer, 2015.

[45] 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.

[46] G. Lowe, "A hierarchy of authentication specifications," in *Computer Security Foundations Workshop*. IEEE, 1997.

[47] 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.

[48] 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.

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

[50] 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.

[51] 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.

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

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

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

[55] 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*, 2013.

[56] 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.

[57] 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.

[58] 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.

Haonan Feng received the B.S. degree in Information Security from Beijing University of Posts and Telecommunications (BUPT) in 2018 and the M.S. degree in Computer Technology from Beijing University of Posts and Telecommunications (BUPT) in 2021. His research interest lies in analyzing security protocol using formal methods. He is a recipient of the special prize of the 4th National Cryptography Competition, China. He is now a security engineer at the Ant Financial Services Group Co., Ltd.

Jingjing Guan received B.S. degree in Information Security from Beijing University of Posts and Telecommunications (BUPT) in 2020. She is currently working toward the M.S. degree in Cyberspace Security in Beijing University of Posts and Telecommunications (BUPT). Her research interest includes formal analysis of security protocols.

Hui Li got her B.S. in Computer Application Technology in 1995 from Jilin University and Ph.D. in Cryptography in 2005 from Beijing University of Posts and Telecommunications (BUPT), China. Now, she is working at the School of Cyberspace Security in BUPT as an associate professor. Her current research interests include security protocol, information security and mobile security.

Xuesong Pan received the B.S. degree in Information Security from Beijing University of Posts and Telecommunications (BUPT) in 2018 and the M.S. degree in Cyberspace Security from Beijing University of Posts and Telecommunications (BUPT) in 2021. His main research interests include protocol security and system security. He is now a software engineer at Huawei Technologies Co., Ltd.

Ziming Zhao is an Assistant Professor at the Department of Computer Science and Engineering (CSE) and the director of the CyberspACe securiTy and forensIcs lab (CactiLab), University at Buffalo. His current research interests include system and software security, trusted execution environment, formal methods for security, and usable security. His research has been supported by the U.S. National Science Foundation (NSF), the U.S. Department of Defense, the U.S. Air Force Office of Scientific Research, and the U.S. National Centers of Academic Excellence in Cybersecurity. He is a recipient of NSF CRII Award. His research outcomes have appeared in IEEE S&P, USENIX Security, ACM CCS, NDSS, ACM MobiSys, ACM TISSEC/TOPS, IEEE TDSC, IEEE TIFS, etc. He is also a recipient of best paper awards from USENIX Security 2019, ACM AsiaCCS 2022, and ACM CODASPY 2014. He received the Ph.D. degree in Computer Science from Arizona State University, Tempe, AZ, in 2014.