A Formal Analysis of the FIDO2 Protocols

Jingjing Guan[1(✉)], Hui Li[1], Haisong Ye[1], and Ziming Zhao[2]

[1] Beijing University of Posts and Telecommunications, Beijing, China
{GuanEr,lihuill,yeeys}@bupt.edu.cn
[2] CactiLab, University at Buffalo, Buffalo, USA
zimingzh@buffalo.edu

Abstract. FIDO2 is the latest member of the Fast IDentity Online (FIDO) protocol suite, which aims at providing unified password-less authentication across the web. We present a formal security analysis of the FIDO2 protocols. We extend the previously presented formalization of the security assumptions and goals of FIDO with FIDO2 specific requirements. We develop a formal model that considers both the CTAP2 and WebAuthn in FIDO2 as a whole. Our formal analysis identifies the minimal security assumptions required for each security goal of FIDO2 to hold. The verification results show FIDO2 fails to achieve some strong authentication properties. The results also reveal that the newly introduced Client PIN mechanism has flaws, and the discovered authenticator rebinding attack and parallel session attacks in UAF still exist in FIDO2.

Keywords: FIDO2 · Formal analysis · Security protocol

1 Introduction

FIDO2 [16,35] is the latest member of the Fast IDentity Online (FIDO) protocol suite. It builds on top of the commercial success of the previous FIDO protocols, including the Universal Second Factor (U2F, a.k.a. CTAP1) [13] and the Universal Authentication Framework (UAF) [20]. FIDO2 aims to support all U2F and UAF use cases and offer a ubiquitous and unified strong authentication across the web. Extended from U2F and UAF, FIDO2 supports user identity verification in 2nd-factor authentication scenarios. By Jan 2022, FIDO2 has been integrated into Android 7.0+ [18], Windows 10 [17], iOS and MacOS [19], Google Chrome, Mozilla Firefox, Microsoft Edge [14], etc.

FIDO2 includes multi-factor and password-less authentication and fixes some discovered flaws in U2F and UAF [32]. Therefore, the protocol architecture and procedures of FIDO2 differ from the counterparts of U2F and UAF significantly. FIDO2 consists of the Client to Authenticator Protocol v2.0 (CTAP2) [16] and the Web Authentication protocol (WebAuthn) [35]. With the Client PIN mechanism in CTAP2, FIDO2 offers user identity verification in 2nd-factor use cases when the authenticator does not have a user interface. WebAuthn provides a standard web API that enables online services to implement FIDO2 authentication into browsers and web platforms.

V. Atluri et al. (Eds.): ESORICS 2022, LNCS 13556, pp. 3–21, 2022.
https://doi.org/10.1007/978-3-031-17143-7_1

To automatically verify the security of UAF, Feng et al. [12] presented a formalization of UAF's security assumptions, goals, and protocol process. Even though FIDO2 and UAF share the same natural language security goals, their semantics and formalization are not the same in the different protocol contexts. For example, the relevant data fields in the security properties need to be adjusted to the messages in FIDO2, and more properties need to be verified in the Client PIN mechanism. In particular, the confidentiality of the key fields, such as the Client PIN provided by the user and the token issued by the authenticator, must be verified. To the best of our knowledge, there is no formal treatment of the two FIDO2 protocols as a whole. Barbosa et al. [3] considered CTAP2 and WebAuthn as independent modules and only analyzed several authentication goals. Guirat et al. [23] only verified WebAuthn and considered privacy goals. Moreover, neither of them considered the scenarios with different authenticator types and transaction authorization modes.

In this paper, we present a formal verification of FIDO2, which shows the security goals of FIDO2 cannot be satisfied in some cases. In particular, we show FIDO2 does not provide stronger security in 2nd-factor scenarios despite its efforts since the Client PIN and token can be exploited to undermine the validity of authentications in registration and authentication processes. The contributions of this paper are as follows:

1. We present a *faithful formal model of the FIDO2*, which considers CTAP2 and WebAuthn as a whole and models scenarios with different authenticator types in registration, authentication, and transaction authorization. We open-source our tool FIDO2Verif[1], which is a front-end to ProVerif [7].
2. We refine the *formalization of FIDO security goals in the context of FIDO2*. In particular, we consider the secrecy of Client PIN and the token issued by the authenticator in FIDO2 due to the newly introduced CTAP2 process. We also formalize several security goals that were not modeled before [12].
3. The verification results show *flaws in FIDO2*. For example, due to the unauthenticated ECDH in CTAP2, FIDO2 fails to achieve the strong authentication property, and the previously discovered authenticator rebinding attack and parallel session attack on UAF are still effective on FIDO2.
4. We present recommendations on how to fix the discovered flaws in FIDO2.

2 Overview of FIDO2

Table 1 presents the acronyms used throughout this paper. CTAP2 is the protocol between the authenticator and the client to share a token, which will be used in the operations of WebAuthn subsequently. There are two operations in WebAuth, namely authenticator registration and authentication. In authenticator registration, users register their certified FIDO2 authenticator with a vendor-signed *attes-*

[1] https://github.com/CactiLab/FIDO2Verif.

Table 1. Acronyms and descriptions.

Acronym	Full Name	Description
newPin	New client PIN	New client PIN that the user enters in FIDO Client when setting or changing the Client PIN
curPin	Current client PIN	Current client PIN the user provides to FIDO Client to change a new Client PIN or apply for a *PinToken*
RpID	Relying party identifier	A domain that identifies the WebAuthn relying party and determines the set of origins on which the public key credential may be exercised
AAGUID	Authenticator attestation globally unique identifier	An identifier that indicates the type (e.g. manufacturer and model) of the authenticator
UHandle	User handle	A value specified by a relying party to map a public key credential to a user account
Chlg	Relying party challenge	A random number provided by the relying party in registration and authentication requests
Tr	Transaction text	Transaction information generated by the relying party that needs to be authorized by the user
TBinding	TLS token binding	A long-lived identifier of TLS bindings spanning multiple TLS sessions and connections from Token Binding protocol
PinToken	Token controlled by client PIN	A random token generated by authenticators and issued to FIDO Client
G	Elliptic-curve parameters	ECC parameters used to establish ECDH shared secret between Authenticator and FIDO Client
K	Shared key	The key established between Authenticator and FIDO Client through ECDH, which is used for symmetric encryption, decryption, and HMAC
CNTR	Signature counter	An integer that increments after each successful authentication
FCData	Client data object	The contextual binding of the WebAuthn relying party and the client
CreID	Credential ID	An identifier to retrieve the credential private key stored in the authenticator or with the credential private key wrapped in
$Cert_{AT}$	Attestation certificate	A certificate for the attestation key pair used by an authenticator to attest to its manufacture and capabilities
sk_{AT}	Attestation private key	The private asymmetric key shared across a large number of FIDO device units made by the same vendor
pk_{AT}	Attestation public key	The public asymmetric key shared across a large number of FIDO device units made by the same vendor
sk_{Cre}	Credential private key	The private key portion of the credential key pair stored in FIDO device or wrapped in *CreID*
pk_{Cre}	Credential public key	The public key portion of the credential key pair, generated by an authenticator and returned to a relying party in registration
k_W	Wrapping key	A key known only to the FIDO device, which is used to encrypt the public key credential source

tation key for the accounts of remote service. The user provides the original credential first, usually a text-based password, and then selects a local authentication mechanism such as swiping a finger, entering a PIN, etc. The authenticator generates a pair of asymmetric *credential keys* and signs the public part with the attestation private key. Then, it sends the credential public key, the signature, and the *attestation certificate* to the server. If the certificate and the signature pass the server's verification, binding between the user's account and the authenticator will be established and recorded in the server and the registration process succeeds. In the authentication stage, the user performs the local authentication method selected before, and the authenticator runs a challenge-response protocol with the server using the *credential private key*.

2.1 Architecture of FIDO2

As shown in Fig. 1, the entities in FIDO2 include a *FIDO Authenticator* (FA), a *FIDO Client* (FC), and a *Relying Party* (RP). FA is a hardware or software cryptographic entity, which can be implemented on-device as platform authenticators, such as biometric or PIN verification modules, or off-device as cross-platform authenticators, such as FIDO Security Keys, mobile devices, wearables, etc. FA stores a vendor-signed attestation private key (sk_{AT}), the attestation certificate ($Cert_{AT}$), and a model identifier ($AAGUID$). FA generates credential key pairs (sk_{Cre}, pk_{Cre}) used in authentication. FAs with persistent storage to store sk_{Cre} are called *client-side storage authenticators*, in which keys are identified by random $CreID$s. *Server-side storage authenticators* are limited in storage capacity. They encrypt the credential private key sk_{Cre} in $CreID$ and send it to the server.

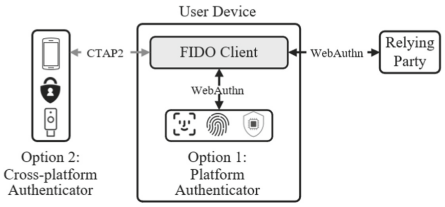

Fig. 1. The FIDO2 architecture

FIDO Client (FC) is implemented in whole or in part of the user agent, e.g., browser. FC stores the identifiers of valid RPs as $xRpID$s. RP consists of a *web server* and a *FIDO server* that utilizes the WebAuthn API to register and authenticate users. In authenticator registration, RP verifies the authenticator and records the binding between the account and FA of the user. The binding will be verified by RP in authentication.

The channel between authenticators and FC can be established through cross-platform transports, such as Bluetooth and NFC, or a platform-specific transport, such as inter-process communication. The communications between cross-platform FA and FC are defined in CTAP2, and the Client PIN mechanism in CTAP2 should be enabled if user identity verification is required in 2nd-factor use cases. FC and RP usually communicate over a TLS channel. The optional steps in FIDO2 include using different types of authenticators, choosing different CTAP2 processes, and using different transaction authorization modes. Different

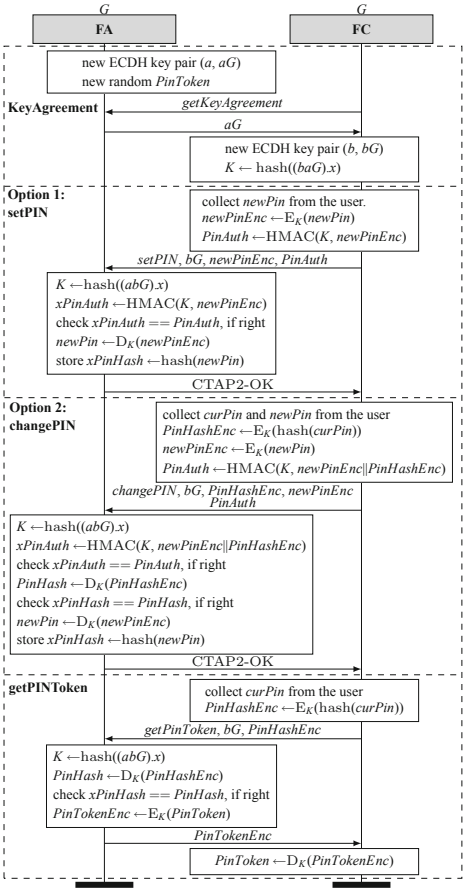

Fig. 2. CTAP2 operations

options result in different protocol processes, which we will explain in detail in the subsequent sections.

2.2 The CTAP2

As shown in Fig. 2, the Client PIN mechanism in CTAP2 consists of the following operations. In *KeyAgreement*, FA and FC negotiate a shared ECDH secret K first, which can be used for symmetric encryption, decryption, and HMAC. K is obtained by hashing the x-coordinate of the point abG on the elliptic curve. The user can choose to initialize or change the Client PIN via the FC on the device through the *setPIN* and *changePIN* process. If the Client PIN has already been set and does not need to be modified, the user enters the current Client PIN on the FC, then the FA and FC directly execute the *getPINToken* process, as long as the FA verifies the Client PIN is correct, it will issue a $PinToken$ to the FC. Within the lifetime of $PinToken$, FC can use this $PinToken$ for the subsequent authenticator registration and authentication with FA without requiring the user to enter the Client PIN again.

2.3 The WebAuthn Protocol

WebAuthn defines the operations of *authenticator registration* and *authentication*, which are shown in Figs. 3 and 4. The contents marked in blue are the operations and message fields when CTAP2 is enabled. We also present the different operations using server-side storage and client-side storage authenticators in this section.

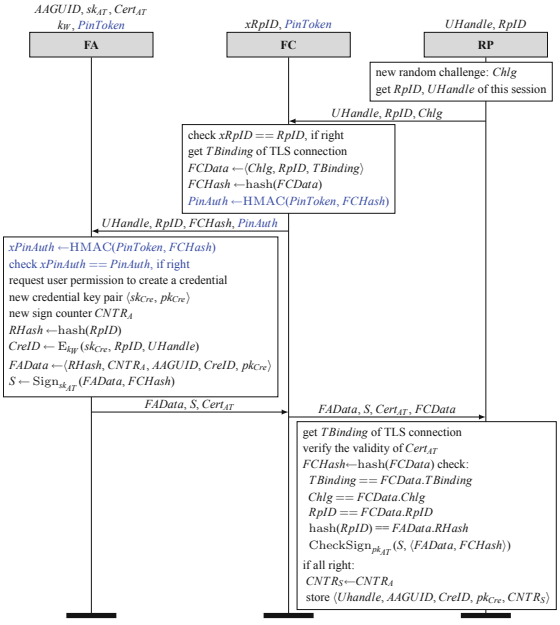

Fig. 3. Authenticator registration (Color figure online)

Authenticator Registration. After the user logs in with the original authentication method, RP generates a registration request and forwards it to FC. FC checks whether $RpID$ is equal to the expected $xRpID$. FIDO2 mitigates the attack caused by a missing check of $AppID$ in U2F (similar to the $RpID$ in FIDO2) [32] and enforces the checking of $RpID$. Subsequently, FC assembles $FCData$ as the contextual bindings of the FIDO session and the TLS session. Then FC calculates $FCHash$, the hash of $FCData$, and $PinAuth$, and forwards the request to FA.

Table 2. Definitions of authenticator types

Authenticator type	Credential storage modality	Authenticator attachment modality	Authentication factor capability
1st factor platform	Client-side	Platform	Multi-factor
1st factor roaming	Client-side	Cross-platform	Multi-factor
2nd factor platform	Server-side	Platform	Single-factor
2nd factor roaming	Server-side	Cross-platform	Single-factor
User-verifying platform	Server-side	Platform	Multi-factor
User-verifying roaming	Server-side	Cross-platform	Multi-factor

FA checks $PinAuth$ and generates a credential key pair $\langle sk_{Cre}, pk_{Cre}\rangle$, and a random signature counter $CNTR_A$ for this account. FA assembles $FAData$, calculates signature S, and returns the response to FC. FC adds $FCData$ to the response from FA and forwards it to RP. RP checks the validity of $Cert_{AT}$, inspects the fields in $FCData$, compares $RHash$ in $FAData$, and verifies the signature S with sk_{AT} to validate the legitimacy of FA.

Authentication. The two transaction confirmation modes of FIDO2 extend the authentication process, and some message fields are added (enclosed with '[]'). The messages in *Generic Transaction Authorization Mode* take the value on the left of '/', and the messages in *Simple Transaction Authorization Mode* take the value on the right of '/'.

RP generates the authentication request and forwards it to FC. After receiving the request, FC checks $RpID$ and constructs $FCData$. Then FC calculates $PinAuth$, the HMAC of $FCHash$ with the $PinToken$ obtained in CTAP2. Finally, FC sends the request to FA. Upon receiving the request, FA checks the $PinAuth$ and asks for the user's permission to perform authentication. Then FA decrypts the $CreID$ with k_W and calculates $RHash$.

In *Generic Transaction Authorization Mode*, $THash$ is included in the response, while in *Simple Transaction Authorization Mode*, the cor-

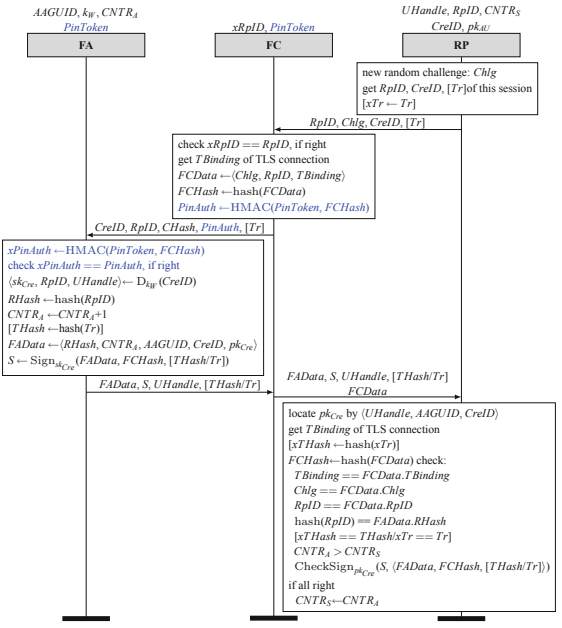

Fig. 4. Authentication (Color figure online)

Table 3. Protocol operations of different types of authenticators

Server-side storage	Reg.	$CreID \leftarrow \mathtt{E}_{k_W}(sk_{Cre},\ RpID,\ UHandle)$
	Auth.	$\langle sk_{Cre},\ RpID,\ UHandle\rangle \leftarrow \mathtt{D}_{k_W}(CreID)$
Client-side storage	Reg.	new random $CreID$ store $\langle CreID,\ sk_{Cre},\ RpID,\ Uhandle,\ CNTR_A\rangle$
	Auth.	get $\langle CreID,\ sk_{Cre},\ RpID,\ Uhandle,\ CNTR_A\rangle$ using $CreID$

responding value is Tr. FA increases the signature counter $CNTR_A$, usually by one, forms the $FAData$, calculates signature S, and sends the response to FC. FC adds $FCData$ to the response from FA and returns it to RP. RP locates the previously stored record by the triple $\langle UHandle, AAGUID, CreID\rangle$, checks the fields in $FCData$, and compares $RHash$ in $FAData$. If the received $THash$ or Tr equals the expected value, the transaction data will be displayed and confirmed by the user. RP verifies the signature S and compares the value of the signature counter. If $CNTR_A$ is greater than $CNTR_S$, RP updates its internally stored $CNTR_S$ with $CNTR_A$.

Operations of Different Types of Authenticators. As shown in Table 2, WebAuthn defines six types of authenticators. Based on how the authentication method is implemented, they can be divided into 1st/2nd/user-verifying authenticators, so as the credential storage capability, access method, and authentication factor. Among them, 1st factor platform authenticators and 1st factor roaming authenticators are client-side storage authenticators, while the other four types of authenticators are server-side storage authenticators. The storage type is determined by where the authentication credential, i.e., sk_{Cre} generated during the authenticator registration phase, is stored. Client-side storage authenticators store the authentication credential sk_{Cre} on the authenticator. As thus, the credential will not leave the user's device and the $CreID$ is a random number that serves as an index to access credentials. A server-side storage authenticator is limited in storage capacity; therefore, it wraps sk_{Cre} in $CreID$ with a wrapping key k_W and sends it to the RP. The different protocol operations of client-side and server-side storage authenticators are shown in Table 3.

3 Formal Verification of FIDO2

The FIDO Security Reference [15] presents informal descriptions of assumptions and security goals in English, which are lengthy and sometimes ambiguous. Because the assumptions in the specifications are strong and often impractical, we refine these descriptions to derive a practical and realistic threat model in our formalization and give formal interpretations of security goals.

3.1 Assumptions and Threat Model

Assumptions on Cryptographic Primitives. We assume cryptographic functions are secure, and attacks on cryptographic algorithms and parameters

Table 4. Formalization of FIDO2 security goals

Type	Goals	Label	Formal description
C.	SG-2	C1∼C8	The wrapping key k_W(C1), the private keys sk_{AT}(C2), sk_{Cre}(C3), the signature counter $CNTR$(C4), the authentication key reference $CreID$(C5), user transaction data Tr(C6), $curPin$ and $newPin$(C7) and $PinToken$(C8) should be secret in FIDO2.
	SG-3		
	SG-8		
	SG-15		
A.	SG-10	A0	**Attack Resistance:** The authentication between entities should be injective agreement on the data fields.
	SG-11		
	SG-12		
	SG-13		
	SG-1	A1	**Strong User Authentication:** The RP must obtain injective agreement on $RpID$, $UHandle$, $AAGUID$, and $CreID$ with the FA after authentication
	SG-14	A2	Transaction Non-Repudiation: The RP must obtain injective agreement on Tr with the authenticator after transaction authorization
	SG-5	A3	**Verifier Leak Resilience:** Supposing that user A has registered on both RP and RP$'$ with the authenticator FA, even if the RP$'$ leaks the information of user A, the RP should still obtain injective agreement on $RpID$, $UHandle$, $AAGUID$, and $CreID$ with the FA after authentication
	SG-6	A4	**Authenticator Leak Resilience:** Supposing that both user A and user B have registered on the RP and RP$'$ with their FA respectively, even if an attacker can steal the information of user B from the FA, the RP should still obtain injective agreement on $RpID$, $UHandle$, $AAGUID$, and $CreID$ with A' FA after authentication
	SG-7	A5	**User Consent:** The RP must obtain injective agreement on $RpID$, $UHandle$, $AAGUID$, and $CreID$ with the FC after registration
	SG-9	A6	**Attestable Properties:** The RP must obtain injective agreement on $RpID$, $UHandle$, $AAGUID$, and pk_{AT} with the authenticator after registration.
P.	SG-4	P1	**Unlinkability:** FIDO2 processes initiated by the same user on different RPs should be observational equivalent to the RPs

are beyond our consideration. **Assumptions on Channels and Entities**. Two types of channels are involved in FIDO2: the channel between FC and FA is established through IPC, while the channel between FC and RP is relying on TLS. Similar to the security assumptions for channels and entities in UAF, we assume that the IPC and TLS channels provide both confidentiality and integrity, hence a Dolev-Yao attacker has no control of messages exchanged over the channel established between honest entities. However, the attackers can initiate a conversation through malicious entities under their control and intercept the information in these sessions. **Assumptions on Data Protections**. We assume the identifiers, including $UHandle$, $AAGUID$, and $RpID$, the public keys pk_{AT} and pk_{AU}, and the elliptic curve parameter G are public. $CreID$, $CNTR$, k_W, or sk_{AT} are not public but can be compromised.

3.2 Security Goals

[SG-1, 4, 5∼7, 9, 10∼14] have been formalized in literature [12], and we formalize [SG-2∼3, 8, 15] for the first time. As shown in Table 4, we formally interpret the goals [SG-1∼15] in the FIDO2 context. We denote the confidentiality, authentication, and privacy goals as 'C.', 'A.', and 'P.', respectively. Because C1∼C6 in

FIDO2 are similar to those of UAF, we focus on **C7** and **C8**, which should be considered because of the newly introduced Client PIN mechanism.

In CTAP2, a shared secret is negotiated between the FA and FC. The user sets the Client PIN on the FC, and then the FC encrypts and forwards it to the FA. Once the Client PIN is leaked, the attacker can initiate the CTAP2 process with a malicious FC and invoke the FA on the user's device. Formally, **C7:** the current Client PIN *curPin* and the newly set Client PIN *newPin* should be secret in the presence of the active attacker during the CTAP2 process.

The CTAP2 process runs before authenticator registration or authentication. After the FA verifies the validity of the Client PIN entered by the user on the FC, the FA will issue a *PinToken* to the FC. Subsequently, the FC needs to use *PinToken* to calculate an HMAC *PinAuth* and add it to the registration or authentication requests sent to the FA. If the verification of *PinAuth* fails, the FA will terminate this operation. Once *PinToken* is leaked, the attacker will be able to forge a request and deceive the user's authenticator to communicate with it. Formally, **C8:** The *PinToken* should be secret in the presence of the active attacker during the CTAP2 process.

4 Formal Models

To formally analyze FIDO2, we use ProVerif [7]. Compared with other commonly used tools, such as Tamarin [30], Scyther [11], AVISPA [1], DEEPSEC [10], CL-AtSe [34], OFMC [5], FDR [29], SATMC [2], Cryptyc [22], ProVerif solved the problem of state explosion under unlimited sessions and provides a user-friendly interface for interactive operation and attack path display. It has been used to analyze multiple security protocols including UAF [12], TLS 1.3 Draft 18 [6], ARINC823 avionic protocols [8], and e-voting protocols [24].

4.1 ProVerif Models of FIDO2

To cover all possible scenarios of FIDO2, we define the following three types of constants: $CTAPType$, $AuType$, and $TrType$, to identify which CTAP2 process the users go through in this protocol, which authenticator is used, and under which authentication mode. The four values, $noCTAP$, $setPIN$, $chgPIN$, and $getToken$, of $CTAPType$ identify the four processes in the CTAP2 process respectively: without enabling the CTAP2 protocol, going through setPIN or changePIN process in CTAP2, and directly perform the getPINToken operation. The values *client* and *server* of $AuType$ identify the scenarios using client or server-side storage FA. The value *empty*, *simple*, and *generic* of $TrType$ are used to distinguish the scenarios of authentication, simple transaction authorization mode, and generic transaction authorization mode. As a result, we analyze 32 different scenarios, including eight scenarios in authenticator registration ($4 \times 2 = 8$ scenarios in total, as the transmission of transaction authorization data is not transmitted in registration), and 24 scenarios in authentication ($4 \times 2 \times 3 = 24$). Each scenario is identified by a specific value of the tuple ($CTAPType$, $AuType$)

or the triple ($CTAPType$, $AuType$, $TrType$), which will determine the branches to go through in this run of the process. When the CTAP2 option is enabled, $PinToken$ will be shared between FA and FC after CTAP2 is completed. The token will be used in subsequent authenticator registration and authentication operations. We model this feature using tables and phases in ProVerif. FA and FC maintain a table for $PinToken$. When the CTAP2 operation is completed, both parties will store the $PinToken$ in the table.

4.2 Verifying Leak Resilience Goals of FIDO2

SG-5 (Verifier Leak Resilience) and SG-6 (Authenticator Leak Resilience) were formalized but not verified in the previous UAF formal analysis [12]. To verify these two goals, we design a scenario with three sets of sessions, which need to be modeled and analyzed separately. User A registers the FA on RP and RP′, while user B registers the FA′ on RP. There are three sets of data after registration: the data between FA and RP, FA and RP′, FA′ and RP. We verify whether the authentication between RP and FA can be satisfied in the case that the FA′ of user B leaks its data registered on RP while RP leaks the data of FA.

5 Security Analysis

To identify the minimal assumptions of each security goal of FIDO2, we develop a tool FIDO2Verif, which is a front-end to ProVerif. Our tool is based on the idea proposed by Basin et al. [4], which was also used in the analysis of multi-factor authentication protocols [26] and the Noise framework [21]. The tool first verifies whether a security goal is satisfied without any assumptions, then increases the number of assumptions until the state of the security property is changed from false to true. If the state of the security properties has not changed after adding all the assumptions about the entity, then we add the assumptions about the data fields. The tool automatically generates 78,336 test cases covering all CTAP2 process options, authenticator types and transaction authorization modes, and the combinations of assumptions on entities and fields.

5.1 Results

Table 5 and Table 6 show the analysis results. ‘Reg.’ means the results of authenticator registration and ‘Auth.’ means the results of authentication. The results show the minimum assumptions required for each security goal in our threat model. ‘✓’ means the goal can be met in all conditions, ‘–’ means not applicable. ‘¬’ before the filed, e.g., ‘$\neg k_W$’, indicates the property only holds when k_W is not revealed. ‘$\neg X[Y]$’ denotes the attackers cannot use their compromised entity X to communicate with entity Y. Since the transaction authorization data Tr can only be transmitted in authentication, and the attestation private key sk_{AT} is only used in registration, there is no verification result for $C6$ in Table 5, and no verification result for $C2$ in Table 6.

Confidentiality Properties. As shown in Table 5, the attackers have no access to k_W and sk_{AT}, since these fields are stored and used only inside FA in authenticator registration. As for client-side storage authenticators, sk_{Cre} is stored inside FA and cannot be obtained by the attacker. While for server-side storage authenticators, sk_{Cre} is encrypted in the $CreID$ by k_W. $CreID$ and $CNTR$ are exposed on the channel as part of the registration response from FA. Therefore, if attackers compromise $C[A]$, they can eavesdrop on the registration response to obtain $CreID$ and $CNTR$. As long as the attacker cannot obtain k_W (the assumption $\neg k_W$ is met) or $CreID$ (the assumption $\neg C[A]$ is met), the secrecy of sk_{Cre} will hold. Since CTAP2 relies on unauthenticated ECDH to negotiate a shared secret between FA and FC, the two participants cannot confirm the validity of their identities. Therefore, if there is a malicious FA′ in the CTAP2 session (breaking assumption $\neg A[C]$), the FA′ can complete the CTAP2 process with the FC and obtain the $curPin$ and $newPin$ entered by the user on FC. In the same way, a malicious FC′ (breaking the assumption $\neg C[A]$) can intercept the $PinToken$ issued by the FA.

Table 5. Verification results in registration.

Reg.	Client-Side	Server-Side
$C1$	–	✓
$C2$	✓	✓
$C3$	✓	$\neg k_W \vee \neg C[A]$
$C4$	$\neg C[A]$	
$C5$		
$C7$	$\neg A[C]$	
$C8$	$\neg C[A]$	
$A5$	$\neg A[C] \wedge \neg C[R]$	
$A6$		
$P1$	✓	✓

Table 6 shows that sk_{Cre} of client-side storage authenticators and k_W of server-side storage authenticators are secure as they are only used within FA in authentication. For server-side storage authenticators, sk_{Cre} is encrypted with k_W as $CreID$, which is part of the authentication request from RP and the authentication response from FA. To protect $CreID$, it is necessary to prevent attackers from obtaining the above messages and the assumptions $\neg A[C]$ (there is no malicious FA to obtain the authentication request relayed from FC) and $\neg C[R]$ (there is no malicious FC to obtain the authentication request sent from RP) and $\neg C[A]$ (there is no malicious FC to obtain the authentication response returned from FA) should be met. As for $CNTR$, which is retrieved from the storage from FA with corresponding $CreID$ and included in the message returned from FA, the assumptions $\neg C[R]$ or $\neg C[A]$ should be met.

In other words, if the attacker cannot get the correct $CreID$ or the authentication response sent by the FA, $CNTR$ will not be intercepted. The assumption required to maintain the confidentiality of Tr in Simple Transaction Authorization Mode is the same as that of $CreID$, which is $\neg A[C]$ and $\neg C[R]$ and $\neg C[A]$. Because both $CreID$ and Tr are included in the request from the RP and the response returned by the FA. To maintain the secrecy of Tr in Generic Transaction Authorization Mode the

Table 6. Verification results in authentication.

Type	Client-Side		Server-Side	
Mode	Simple	Generic	Simple	Generic
$C1$	–		✓	
$C3$	✓		$\neg k_W \vee (\neg A[C] \wedge \neg C[R] \wedge \neg C[A])$	
$C4$	$\neg C[R] \vee \neg C[A]$			
$C5$	$\neg A[C] \wedge \neg C[R] \wedge \neg C[A]$			
$C6$	$(CreID)$	$\neg A[C] \wedge \neg C[R]$	$(CreID)$	$\neg A[C] \wedge \neg C[R]$
$C7$	$\neg A[C]$			
$C8$	$\neg C[A]$			
$A1$	$\neg C[R] \vee \neg C[A]$			
$A2$				
$A3$	✓		✓	
$A4$				
$P1$	✓		✓	

protocol should satisfy $\neg A[C]$ and $\neg C[R]$ as Tr is only involved in the request from the RP.

Authentication Properties. As shown in Table 5, to achieve the authentication goals in authenticator registration, the minimal assumption $\neg A[C]$ and $\neg C[R]$ should be met. Whether sk_{AT} is leaked or not has little effect on the authentication in authenticator registration. RP verifies the validity of FA by checking the signature S, which is signed by sk_{AT}, and inspects the fields in $FCData$ to confirm the binding relationship between a FIDO2 session and a TLS session. As the same sk_{AT} is manufactured in a batch, the attacker can simply use an FA$'$ from the same batch as the user's FA. Once $\neg A[C]$ and $\neg C[R]$ is violated, the attackers can use their FA$'$ to generate a valid response and forward it to the FC on the user's device with a compromised FC$'$. The response will pass the inspection of RP and be considered a legitimate response. Thereafter, the user's account will be bound to the FA$'$ held by the attacker instead of the FA of the user, and the user is not aware of it.

After successful registration, the binding between the user account and the FA has been established and RP has recorded the binding as $\langle UHandle, AAGUID, CreID, pk_{Cre}, CNTR_S\rangle$. RP verifies the signature S with pk_{Cre} and checks the value of $CNTR$. Since the private key sk_{Cre} and the counter $CNTR$ are stored in the user's FA, which can only be retrieved with the corresponding $CreID$ in the authentication request. If the assumption $\neg C[A]$ or $\neg C[R]$ is not satisfied, the attacker will be able to intercept $CreID$ an authentication request with a malicious FC, with the correct $CreID$, the malicious FC can receive a valid authentication response from the user's FA and forward it to RP.

Our analysis results show that there is no data leakage on RP or FA, i.e., the authentication goals $A3$ and $A4$ are satisfied. Even if the RP leaks user B's data, it still does not affect the RP's authentication of other users. For the same user, even if the user's authentication data of an RP$'$ is leaked, the user's authentication on other RPs can still be guaranteed. The RP authenticates the user by verifying the binding relationship of the triple: the user account (identified by $UHandle$), the authenticator (identified by $AAGUID$), and the RP (identified by $RpID$), which is associated with the asymmetric key pair (sk_{Cre}, pk_{Cre}) generated by FA. The public key pk_{Cre} will be sent to RP, while FA saves the private key sk_{Cre}. As long as the private key bound between user A and RP is not leaked, even if RP$'$ leaks user A's information, including the public key of user A on RP$'$, or user B who has registered on the same RP leaks the private key, the authentication of user A from the RP will not be affected.

Privacy Properties. Our results show that FIDO2 satisfies unlinkability in authenticator registration and authentication. $\langle RHash, CNTR_A, AAGUID, CreID, pk_{Cre}, S, Cert_{AT}, FCData\rangle$ No field in the registration response is associated with the user. As both $AAGUID$ and the corresponding $Cert_{AT}$ are shared by a large batch of authenticators, it is difficult, even impossible for the

RP to locate the sessions of the same $AAGUID$ and $Cert_{AT}$ to a single user. $CNTR_A$ records the number of times of authentication for a single user at the specific RP, but cannot directly link the session between different RPs.

RP receives the authentication response $\langle RHash,\ CNTR_A,\ AAGUID,\ CreID,\ pk_{Cre},\ S,\ UHandle,\ [THash\ /Tr],\ FCData\rangle$. Note that it is pointed out in Sect. 4 of WebAuthn [35]: "$UHandle$ should be an opaque byte sequence and not contain any personally identifying information about the user". The RPs cannot infer the user's personal identification information from any instance of $UHandle$. And $UHandle$ is the user identifier specified by an RP that can be used to map the conversations to a specific user only within this RP but different RPs cannot link the session to the specific user with $UHandle$. $\langle CNTR_A,\ CreID$, and $pk_{Cre}\rangle$ are unique for each binding $\langle RpID,\ AAGUID,\ UHandle\rangle$ and cannot be used to distinguish user sessions across different RPs.

5.2 Attacks

MITM Attack in CTAP2. FIDO2 introduces the Client PIN mechanism to improve the security in 2nd-factor authentication scenarios. However, the verification results show that this goal cannot be achieved. Since FA and FC negotiate a shared secret with unauthenticated ECDH, FA and FC cannot establish any trust through this process, making it vulnerable to man-in-the-middle (MITM) attacks. In CTAP2, if the assumption $\neg A[C]$ is not satisfied, the attacker can obtain the $curPin$ and $newPin$. If the assumption $\neg C[A]$ is not satisfied, the attacker can intercept the $PinToken$. The process of MITM Attack in CTAP2 is shown in Fig. 5. The attacker forwards the getKeyAgreement request initiated by FC, intercepts the ECDH public key aG sent by FA, replaces it with the cG generated by itself, and forwards it to FC. The shared secret negotiated between the FA and the attacker is caG, and the attacker can derive a shared key K_1 with FA. The FC generates the ECDH key pair $(b,\ bG)$, and uses the secret cbG shared with the attacker to derive the shared key K_2. After collecting the $newPin$ from the user, the FC encrypts it with K_2 and sends the message $\langle bG,\ newPinEnc\rangle$ to FA. After receiving the message from FC containing the ECDH public key bG, attackers can replace it with their public key cG and send it to FA. Therefore, the attacker negotiates shared secrets caG and cbG with FA and FC respectively, and then derives a shared key K_1 with FA and a shared key K_2 with FC. After that, the attacker can decrypt the $newPinEnc$ and $curPinEnc$ sent by FC, and the $PinTokenEnc$ from FA to obtain the plaintext $newPin$, $curPin$, and $PinToken$ respectively.

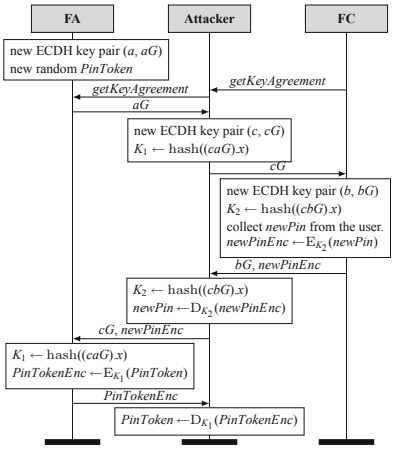

Fig. 5. MITM Attack in CTAP2

Authenticator Rebinding Attack. In registration, if the assumptions $\neg A[C]$ and $\neg C[R]$ are not satisfied, the attackers can implement the authenticator rebinding attack to bind the victim's account to the FA$'$ under their control. We consider an attacker who has the same model of authenticator as the user (with the same sk_{AT} in it) and can compromise the FC on the victim's device (to get access to the channel CA and CR). We denote the FIDO Client on the victim's device as client FC and the FIDO Client on the adversary's device as client FC$'$. The adversaries can bind the victim's account to their authenticators through the following steps: 1) The victim initiates a registration, and the malicious FC on the victim's device establishes a connection with the RP to obtain the registration request; 2) FC redirects the request to FC$'$; 3) FC$'$ continue the FIDO2 registration operations with the authenticator to bind the victim's account with FA$'$ held by the attacker; 4) The FC pretends to complete subsequent operations with the user, making the user believe the binding was successful; 5) The FC$'$ forwards the response to FC; 6) FC sends the response to the RP to complete subsequent operations in registration. In the operations above, $FCData$ is generated by the FC on the victim's device, and the authenticator used by the attacker has the same sk_{AT} as the user's, the returned registration response can be successfully verified by the RP and the user account is bound to the attacker's authenticator. Thereafter, the attackers can log in to the victim's account with their credential and bypass the biometric verification in FIDO2.

Parallel Session Attack. This attack breaks the authentication properties if the assumptions $\neg C[A]$ or $\neg C[R]$ are not satisfied. RP accepted an authentication response after verifying the signature S signed by sk_{Cre} and checking $CNTR_A$. After successful registration, sk_{Cre} and $CNTR_A$ are stored inside the FA and the response with the correct S and $CNTR_A$ will be returned only if A receives a correct $CreID$. With the following steps, the attacker does not need to compromise the FA of the user to obtain sk_{Cre} or k_W, but tamper with FC and initiates a "parallel session" to impersonate a legitimate user.

We assume users have installed a malicious FC on their devices. The attacker can implement the parallel session attack with the following steps: 1) Once a victim launches an authentication session, the attackers can be informed by the malicious FC and initiated their own session on FC$'$ with the victim's identifier $UHandle$ to the same RP to get the correct $CreID$; 2) FC$'$ sends the request generated by itself to client FC. $FCHash$ is the hash of $FCData$ generated from the connection established between FC$'$ and the RP; 3) FC forwards the request to the FA on the victim's device and obtains the response. As $CreID$ in the request corresponds to the $UHandle$ of the victim, FA can successfully retrieve the sk_{Cre} to generate the signature S and get the $CNTR$; 4) FC redirects the response to FC$'$ and FC$'$ appends its own $FCData$ to the response; 5) FC$'$ returns the response to the RP to complete the authentication.

Since the $FCData$ in the response corresponds to the connection established between FC' and the RP and the signature is, in effect, generated by the victim's

authenticator with correct sk_{Cre}, the attackers will pass the verifications in RP and gain access to the victim's account on their own device.

Privacy Disclosure Attack. Attackers can intercept the user's personal data with a malicious FC. In registration, if the assumption $\neg C[A]$ is not satisfied, the original $CNTR$ will be exposed to attackers in the registration response from FA. Later in authentication, if the protocol does not satisfy the assumption $\neg C[R] \vee \neg C[A]$, attackers can get the increased $CNTR$ to estimate the number of times of the completed authentication and infer user behavior based on this. Additionally, with the malicious FC, attackers can also distinguish whether the user is performing authentication or transaction authorization by checking whether the field Tr is involved in the response from RP. Thereafter, attackers can also intercept the victim's transaction authorization text Tr.

5.3 Recommendations

Explicit Definitions of Threat Model. We suggest that the standard explicitly requires the assumptions of channel and component compromise. A clearly specified threat model can provide constructive guidance for the design and implementation of the protocol. On the contrary, an ambiguous threat model may introduce problems in the design process and even introduce vulnerabilities in potential practical implementations. Our analysis results show that the security of FIDO2 relies on the security of the channel between entities and the secure storage module inside the entities. However, there is a lack of a clear definition of entity and channel compromise scenarios in the specifications. Therefore, the compromise scenarios should be considered in the protocol design, and the recommended implementation of the secure channel and secure storage should be clearly given in the specifications.

Enhancing the security of CTAP2. We suggest enhancing the authentication of FA and FC in the CTAP2 process. The current design of CTAP2 relies on unauthenticated ECDH to negotiate the shared secret between FA and FC, making it vulnerable to man-in-the-middle attacks. The attacker can manipulate the shared secret between FA and FC and then decrypt the ciphertext message transmitted between FA and FC, while the user is unaware of it. It is necessary to add the verification of the validity of FA and FC in CTAP2, and confirm that the FA and FC in the CTAP2 session are the same FA and FC used in authenticator registration or authentication subsequently.

Authenticating FC at RP. We suggest adding an authentication mechanism for FC. In particular, for some services with high security requirements, such as financial transactions, it is necessary to enforce the authentication of FC. The authenticated channel between FC and RP is usually established by TLS. However, client-side authentication is only optional in TLS. As there is no attestation

or authentication mechanism of FC involved in FIDO2, there may be malicious FCs participating in the communication. Some security properties cannot hold if channel CA and CR are compromised.

Protecting User's PII. We suggest that some concealment mechanism should be applied to $CNTR$ and Tr before sending them on the channel. Based on the existing protocol process, with the encapsulation mechanism of Elliptic Curve Integrated Encryption Scheme (ECIES) [33] used to improve the privacy of 5G AKA [36], $CNTR$ and Tr be encrypted before sending on the channel. RP generates an ephemeral ECC private-public key pair (r, R) for each session, such that $R = rG$, and adds the public portion R to the registration or authentication request. Then RP generate a symmetric session key $k_s = KDF(r, PK)$, $PK = pk_{AT}$ in authenticator registration and $PK = pk_{Cre}$ in authentication. After receiving the request, FA can derive the corresponding session key $k_a = KDF(R, SK)$, $SK = sk_{AT}$ in authenticator registration and $SK = sk_{Cre}$ in authentication. Both parties can use the symmetric keys k_s and k_a for concealment and de-concealment. The FC between FA and RP has no access to the private key, r, sk_{AT}, or sk_{Cre}, and cannot derive correct k_s and k_a. Therefore, the attacker cannot intercept $CNTR$ and Tr by simply compromising the FC.

6 Related Work

Both manual analysis [9,25,27,28,31] and formal methods [12,26,32] have been applied to verify the security of FIDO protocols. Feng et al. provided a faithful formal model of the UAF protocol considering the use case with different authenticator types and various optional steps and their analysis covers most of the security properties in UAF specifications. Their verification results confirmed previously found vulnerabilities and disclosed new attacks which can be exploited in real-world apps [12]. Pereira et al. formally modeled U2F but their oversimplified threat model only considered the option of checking $AppID$, which failed to find out more underlying vulnerabilities [32]. Jacomme et al. defined a fine-grained threat model considering different scenarios with the combinations of malware, phishing, and TLS fingerprint spoofing in U2F but they mainly focused on the authentication goals, ignoring the confidentiality properties in U2F. They did a simple verification of unlinkability between two accounts on two different authenticators and the two accounts on the same authenticator, which is different from the unlinkability specified in the standard. And they did not verify the unlinkability in the registration phase. [26].

There have been several efforts to formally verify FIDO2. Guirat et al. focused on formally analyzing WebAuth. They only presented the verification results of privacy properties and lacked the description and verification of confidentiality and authentication properties [23]. Barbosa et al. attempted to formally analyze FIDO2 in computational model and conducted a modular analysis of CTAP2 and WebAuthn separately. However, they did not consider the optional

steps in FIDO2, including using different types of authenticators, choosing different CTAP2 options, and different transaction authorization modes, nor does it analyze the unlinkability properties in the specifications [3]. Different from the previous work, we provide a formal model of FIDO2 that considers CTAP2 and WebAuthn as a whole and covers scenarios with different types of authenticators and transaction authorization modes. We formally describe all the security goals mentioned in the FIDO2 specifications, including confidentiality, authentication, and privacy properties, and analyze these goals in all the above scenarios.

7 Conclusion

In this paper, we formally analyze FIDO2, the latest member of the FIDO protocol suite. We provide a detailed analysis of the specifications to formally interpret the security assumptions and goals and offer a faithful formal model of FIDO2. Our model is substantially more detailed than those of previous work, as the model views CTAP2 and WebAuthn as a whole and can cover the scenarios using different types of authenticators in different authentication modes. We use the ProVerif tool to automate the analysis of FIDO2 in symbolic model and identified the minimal assumptions required for each property. Our analysis in ProVerif shows that FIDO2 still fails to achieve the strong authentication property in some cases and the attacks previously discovered in UAF still exist in FIDO2. We also present several concrete recommendations to fix the issues in FIDO2. In future work, we plan to make improvements to the issues found in the FIDO protocol suits to defend against the attacks found in this paper and to formally verify the improved version of the protocol.

Acknowledgements. The research of Beijing University and Posts and Telecommunications was supported 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. Armando, A., et al.: The AVISPA tool for the automated validation of internet security protocols and applications. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 281–285. Springer, Heidelberg (2005). https://doi.org/10.1007/11513988_27
2. Armando, A., Carbone, R., Compagna, L.: SATMC: a sat-based model checker for security-critical systems. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 31–45. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_3
3. Barbosa, M., Boldyreva, A., Chen, S., Warinschi, B.: Provable security analysis of FIDO2. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12827, pp. 125–156. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84252-9_5
4. Basin, D., Cremers, C.: Know your enemy: compromising adversaries in protocol analysis. ACM Trans. Inf. Syst. Secur. (TISSEC) **17**(2), 1–31 (2014)

5. Basin, D., Mödersheim, S., Viganò, L.: OFMC: a symbolic model checker for security protocols. Int. J. Inf. Secur. **4**(3), 181–208 (2004). https://doi.org/10.1007/s10207-004-0055-7
6. Bhargavan, K., Blanchet, B., Kobeissi, N.: Verified models and reference implementations for the TLS 1.3 standard candidate. In: IEEE Symposium on Security and Privacy (S&P), pp. 483–502 (2017)
7. Blanchet, B.: Modeling and verifying security protocols with the applied pi calculus and ProVerif. Found. Trends Priv. Secur. **1**(1–2), 1–135 (2016)
8. Blanchet, B.: Symbolic and computational mechanized verification of the ARINC823 avionic protocols. In: IEEE Computer Security Foundations Symposium (CSF), pp. 68–82 (2017)
9. Chang, D., Mishra, S., Sanadhya, S.K., Singh, A.P.: On making U2F protocol leakage-resilient via re-keying. IACR Cryptol. ePrint Arch. **2017**, 721 (2017)
10. Cheval, V., Kremer, S., Rakotonirina, I.: DEEPSEC: deciding equivalence properties in security protocols theory and practice. In: IEEE Symposium on Security and Privacy (S&P), pp. 529–546 (2018)
11. Cremers, C.J.: Unbounded verification, falsification, and characterization of security protocols by pattern refinement. In: ACM Conference on Computer And Communications Security (CCS), pp. 119–128 (2008)
12. Feng, H., Li, H., Pan, X., Zhao, Z., Cactilab, T.: A formal analysis of the FIDO UAF protocol. In: Network and Distributed Systems Security Symposium (NDSS), pp. 1–15 (2021)
13. FIDO Alliance: Universal 2nd factor U2F overview (2017). https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-u2f-overview-v1.2-ps-20170411.html
14. FIDO Alliance: CNET: password-free web security is coming to Chrome, Firefox, Edge (2018). https://fidoalliance.org/cnet-password-free-web-security-is-coming-to-chrome-firefox-edge/
15. FIDO Alliance: FIDO security reference 2018). https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-security-ref-v2.0-id-20180227.html
16. FIDO Alliance: Client to authenticator protocol (CTAP) - proposed standard (2019). https://fidoalliance.org/specs/fido-v2.0-ps-20190130/fido-client-to-authenticator-protocol-v2.0-ps-20190130.html
17. FIDO Alliance: Microsoft achieves FIDO2 certification for Windows Hello (2019). https://fidoalliance.org/microsoft-achieves-fido2-certification-for-windows-hello
18. FIDO Alliance: News: your Google Android 7+ phone is now a FIDO2 security key (2019). https://fidoalliance.org/news-your-google-android-7-phone-is-now-a-fido2-security-key
19. FIDO Alliance: Expanded support for FIDO authentication in iOS and MacOS (Jul 2020), https://fidoalliance.org/expanded-support-for-fido-authentication-in-ios-and-macos
20. FIDO Alliance: FIDO UAF protocol specification (2020). https://fidoalliance.org/specs/fido-uaf-v1.2-ps-20201020/fido-uaf-protocol-v1.2-ps-20201020.html
21. Girol, G., Hirschi, L., Sasse, R., Jackson, D., Cremers, C., Basin, D.: A spectral analysis of noise: a comprehensive, automated, formal analysis of Diffie-Hellman protocols. In: USENIX Security Symposium (2020)
22. Gordon, A.D., Jeffrey, A.: Types and effects for asymmetric cryptographic protocols. J. Comput Secur. (JCS) **12**(3–4), 435–483 (2004)
23. Guirat, I.B., Halpin, H.: Formal verification of the W3C web authentication protocol. In: Annual Symposium and Bootcamp on Hot Topics in the Science of Security (HoTSoS), pp. 1–10 (2018)

24. Hirschi, L., Cremers, C.: Improving automated symbolic analysis of ballot secrecy for e-voting protocols: A method based on sufficient conditions. In: IEEE European Symposium on Security and Privacy (EuroS&P), pp. 635–650 (2019)
25. Hu, K., Zhang, Z.: Security analysis of an attractive online authentication standard: FIDO UAF protocol. Chin. Commun. **13**(12), 189–198 (2016)
26. Jacomme, C., Kremer, S.: An extensive formal analysis of multi-factor authentication protocols. ACM Trans. Priv. Secur. (TOPS) **24**(2), 1–34 (2021)
27. Leoutsarakos, N.: What's wrong with FIDO? https://zeropasswords.com/pdfs/WHATisWRONG_FIDO.pdf (2011)
28. Loutfi, I., Jøsang, A.: FIDO trust requirements. In: Buchegger, S., Dam, M. (eds.) Secure IT Systems. LNCS, vol 9417, pp. 139–155. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26502-5_10
29. Lowe, G.: Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In: Margaria, T., Steffen, B. (eds.) TACAS 1996. LNCS, vol. 1055, pp. 147–166. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61042-1_43
30. Meier, S., Schmidt, B., Cremers, C., Basin, D.: The TAMARIN prover for the symbolic analysis of security protocols. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 696–701. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_48
31. Panos, C., Malliaros, S., Ntantogian, C., Panou, A., Xenakis, C.: A security evaluation of Fido's uaf protocol in mobile and embedded devices. In: Piva, A., Tinnirello, I., Morosi, S. (eds.) TIWDC 2017. CCIS, vol. 766, pp. 127–142. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67639-5_11
32. Pereira, O., Rochet, F., Wiedling, C.: Formal analysis of the FIDO 1.x protocol. In: Imine, A., Fernandez, J.M., Marion, J.-Y., Logrippo, L., Garcia-Alfaro, J. (eds.) FPS 2017. LNCS, vol. 10723, pp. 68–82. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75650-9_5
33. Standards for Efficient Cryptography Group: SEC 1: Elliptic curve cryptography version 2.0, standards for efficient cryptography (2009). https://www.secg.org/sec1-v2.pdf
34. Turuani, M.: The CL-Atse protocol analyser. In: International Conference on Rewriting Techniques and Applications (RTA). pp. 277–286. Springer (2006)
35. W3C: Web authentication: An API for accessing public key credentials level 2 (2021). https://www.w3.org/TR/webauthn-2/
36. Wang, Y., Zhang, Z., Xie, Y.: Privacy-preserving and standard-compatible AKA protocol for 5G. In: USENIX Security Symposium, pp. 3595–3612 (2021)