Provable Security Analysis of FIDO2

Manuel Barbosa[1], Alexandra Boldyreva[2], Shan Chen[2], and Bogdan Warinschi[3]

[1]University of Porto
mbb@fc.up.pt
[2]Georgia Institute of Technology
sasha@gatech.edu dragoncs16@gmail.com
[3]University of Bristol & Dfinity
bogdan.warinschi@gmail.com

Abstract

We carry out the first provable security analysis of the new FIDO2 protocols, the promising FIDO Alliance's proposal for a standard for *passwordless* user authentication. Our analysis covers the core components of FIDO2: the W3C's Web Authentication (WebAuthn) specification and the new Client-to-Authenticator Protocol (CTAP2).

Our analysis is *modular*. For WebAuthn and CTAP2, in turn, we propose appropriate security models that aim to capture their intended security goals and use the models to analyze their security. First, our proof confirms the authentication security of WebAuthn. Then, we show CTAP2 can only be proved secure in a weak sense; meanwhile we identify a series of its design flaws and provide suggestions for improvement. To withstand stronger yet realistic adversaries, we propose a generic protocol called sPACA and prove its strong security; with proper instantiations sPACA is also more efficient than CTAP2. Finally, we analyze the overall security guarantees provided by FIDO2 and WebAuthn+sPACA based on the security of its components.

We expect that our models and provable security results will help clarify the security guarantees of the FIDO2 protocols. In addition, we advocate the adoption of our sPACA protocol as a substitute of CTAP2 for both stronger security and better performance.

1 Introduction

Motivation. Passwords are pervasive yet insecure. According to some studies, the average consumer of McAfee has 23 online accounts that require a password [18], and the average employee using LastPass has to manage 191 passwords [7]. Not only are the passwords difficult to keep track of, but it is well-known that achieving strong security while relying on passwords is quite difficult (if not impossible). According to the Verizon Data Breach Investigations Report, 81% of hacking-related breaches relied on either stolen and/or weak passwords [35]. And what some users may consider an acceptable password, may not withstand the sophisticated and powerful modern password cracking tools. Moreover, even strong passwords may fall prey to phishing attacks and identity fraud. According to Symantec, in 2017, phishing emails were the most widely used means of infection, employed by 71% of the groups that staged cyber attacks [30].

An ambitious project which tackles the above problem is spearheaded by the Fast Identity Online (FIDO) Alliance. A truly international effort, the alliance has working groups in the US, China, Europe, Japan, Korea and India and has brought together many companies and types of vendors, including Amazon, Google, Microsoft, Apple, RSA, Intel, Yubico, Visa, Samsung, major banks, etc.

The goal is to enable user-friendly passwordless authentication secure against phishing and identity fraud. The core idea is to rely on security devices (controlled via biometrics and/or PINs) which can

then be used to register and later seamlessly authenticate to online services. The various standards defined by FIDO formalize several protocols, most notably Universal Authentication Framework (UAF), the Universal Second Factor (U2F) protocols and the new FIDO2 protocols: W3C's Web Authentication (WebAuthn) and FIDO Alliance's Client-to-Authenticator Protocol v2.0 (CTAP2[1]).

FIDO2 is moving towards wide deployment and standardization with great success. Major web browsers including Google Chrome and Mozilla Firefox have implemented WebAuthn. In 2018, Client-to-Authenticator Protocol (CTAP)[2] was recognized as international standards by the International Telecommunication Union's Telecommunication Standardization Sector (ITU-T). In 2019, WebAuthn became an official web standard. Also, Android and Windows Hello earned FIDO2 Certification.

Although the above deployment is backed-up by highly detailed description of the security goals and a variety of possible attacks and countermeasures, these are informal [21]. Our work aims to establish the exact security guarantees of the FIDO2 protocols and to identify and fix potential design flaws and vulnerabilities that hinder their widespread use.

Related Work. Some initial work in this direction already exists. Hu and Zhang [24] analyzed the security of FIDO UAF 1.0 and identified several vulnerabilities in different attack scenarios. Later, Panos *et al.* [32] analyzed FIDO UAF 1.1 and explored some potential attack vectors and vulnerabilities. However, both works were informal. FIDO U2F and WebAuthn were analyzed using the applied pi-calculus and ProVerif tool [22,26,33]. Regarding an older version of FIDO U2F, Pereira *et al.* [33] presented a server-in-the-middle attack and Jacomme and Kremer [26] further analyzed it with a structured and fine-grained threat model for malware. Guirat and Halpin [22] confirmed the authentication security provided by WebAuthn while pointed out that the claimed privacy properties (i.e., account unlinkability) failed to hold due to the same attestation key pair used for different servers.

However, *none* of the existing work employs the cryptographic provable security approach to the FIDO2 protocols in the course of deployment. In particular, there is no analysis of CTAP2, and the results for WebAuthn [26] are limited in scope: as noted by the authors themselves, their model "makes a number of simplifications and so much work is needed to formally model the complete protocol as given in the W3C specification". The analysis in [26] further uses the *symbolic* model (often called the Dolev-Yao model [19]), which captures weaker adversarial capabilities than those in computational models (e.g., the Bellare-Rogaway model [12]) employed by the provable security approach we adopt here.

The works on two-factor authentication (e.g., [17,28]) are related to our work, but the user in such protocols has to use the password *and* the second-factor device during each authentication/login. With FIDO2, *there is no password* during the user registration or authentication phases. The PIN used in FIDO2 is meant to authorize a browser (called a client) access to the authentication token; the server does not use passwords at all.[3] Some two-factor protocols can also generate a binding cookie after the first login to avoid using the two-factor device or even the password for future logins. However, this requires trusting the browser, i.e., a malicious browser can log in as the user without having the two-factor device (or the password). FIDO2 uses the PIN to prevent an attacker with a stolen device from authenticating to a server from a new browser.

Our security analysis is not directly applicable to federated authentication protocols such as Kerberos, OAuth, or OpenID. FIDO2 allows the user to keep a single hardware token that it can use to authenticate to multiple servers without having to use a federated identity. The only trust anchor is an attestation key pair for the token. To the best of our knowledge, there are no complete and formal security models for federated authentication in the literature, but such models would differ significantly from the ones we consider here. It is interesting to see how FIDO2 and federated authentication can be used securely together; we leave this as an interesting direction for future work. Our

[1] The older version is called CTAP1/U2F.

[2] CTAP refers to both versions: CTAP1/U2F and CTAP2.

[3] Some form of prior user authentication method is required for registration of a new credential, but this is a set-up assumption for the protocol.

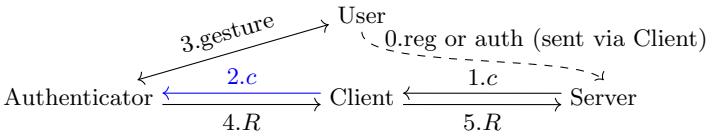

Figure 1: FIDO2 flow (simplified): blue = CTAP2 authorized message.

model could, however, be adapted to analyze some second-factor authentication protocols like Google 2-step [2].

Our Focus. Our work provides the first provable security analysis of the latest FIDO2 protocols. Our focus is to clarify the formal trust model assumed by the protocols and its impact on practical security guarantees and potential vulnerabilities. The analysis covers the actions of human users authorizing the use of credentials via *gestures* and shows that, depending on the capabilities of security devices, such gestures enhance the security of FIDO2 protocols in different ways. We concentrate on the FIDO2 authentication properties and leave the study of its less central anonymity goals for future work.[4]

FIDO2 Overview. FIDO2 consists of two core components (see Figure 1 for the simplified flow). WebAuthn is a web API that can be built into browsers to enable web applications to integrate user authentication. At its heart, WebAuthn is a *passwordless* "challenge-response" scheme between a server and a user. The user relies on a trusted authenticator device (e.g., a security token or a smartphone) and a possibly untrusted client (e.g., a browser or an operating system installed on the user's laptop). Such a device-assisted "challenge-response" scheme works as follows (details in Section 5). First, in the registration phase, the server sends a random challenge to the security device through the client. In this phase, the device signs the challenge using its long-term embedded attestation private key, along with a new public key credential to use in future interactions; the credential is included in the response to the server. In the subsequent interactions, which correspond to user authentication, the challenge sent by the server is signed by the device using the secret key corresponding to the credential. In both cases, the signature is verified by the server.

The other FIDO2 component, CTAP2, specifies the communication between an authenticator device and the client (usually a browser). Its goal is to guarantee that the client can only use the authenticator device with the user's permission, which the user gives by 1) entering a PIN when the device powers up and 2) directly using the device interface (e.g., a simple push-button) to authorize registration and authentication operations. CTAP2 specifies how to configure an authenticator with a user's PIN. Roughly speaking, its security goal is to "bind" a trusted client to the set-up authenticator by requiring the user to provide the correct PIN, such that the authenticator accepts only messages sent from a "bound" client. We remark that, surprisingly, CTAP2 relies on the (unauthenticated) Diffie-Hellman key exchange. The details are in Section 7.

Our Contributions. We perform the first thorough cryptographic analysis of the authentication properties guaranteed by FIDO2 using the provable security approach. Our analysis is conducted in a *modular* way. That is, we first analyze WebAuthn and CTAP2 components separately and then derive the overall security of a typical use of FIDO2.

Provable security of WebAuthn. We start our analysis with the simpler base protocol, WebAuthn. We define the class of *passwordless authentication (PlA)* protocols which aims to capture the syntax and security of the WebAuthn protocol. Our PlA model considers an authenticator and a server (often referred to as a relying party) communicating through a client, which consists of two phases. The server is assumed to know the attestation public key that uniquely identifies the authenticator. In the *registration* phase the authenticator and the server communicate with the intention to establish some joint state corresponding to this registration session: this joint state fixes a credential, which is

[4]Intuitively, anonymity should hold because a FIDO2 user has no associated identity and different users using the same set of tokens (that share the same attestation key pair) should look indistinguishable to the server.

bound to the authenticator's attestation public key vk and a server identity id_S (e.g., a server domain name). The server gets the guarantee that the joint state is stored in a specific authenticator, which is assumed to be tamper-proof. The joint state can then be used in the *authentication* phase. Here, the authenticator and the server engage in a message exchange where the goal of the server is to verify that it is interacting with the same authenticator that registered the credential bound to (vk, id_S).

Roughly speaking, a PlA protocol is secure if, whenever an authentication/registration session completes on the server side, there is a unique partnered registration/authentication session which completed successfully on the authenticator side. For authentication sessions, we further impose that there is a unique associated registration session on both sides, and that these registration sessions are also uniquely partnered. This guarantees that registration contexts (i.e., the credentials) are *isolated* from one another; moreover, if a server session completes an authentication session with an authenticator, then the authenticator must have completed a registration session with the server earlier. We use the model thus developed to prove the security of WebAuthn under the assumption that the underlying hash function is collision-resistant and the signature scheme is unforgeable. Full details can be found in Section 5.

Provable security of CTAP2. Next we study the more complex CTAP2 protocol. We define the class of *PIN-based access control for authenticators (PACA)* protocols to formalize the general syntax of CTAP2. Although CTAP2 by its name may suggest a two-party protocol, our PACA model involves the user as an additional participant and therefore captures human interactions with the client and the authenticator (e.g., the user typing its PIN into the browser window or rebooting the authenticator). A PACA protocol runs in three phases as follows. First, in the authenticator setup phase, the user "embeds" its PIN into the authenticator via a client and, as a result, the authenticator stores a PIN-related long-term state. Then, in the binding phase, the user authorizes the client to "bind" itself to the authenticator (using the same PIN). At the end of this phase, the client and the authenticator end up with a (perhaps different) binding state. Finally, in the access channel phase, the client is able to send any authorized message (computed using its binding state) to the authenticator, which verifies it using its own binding state. Note that the final established access channel is *unidirectional*, i.e., it only guarantees authorized access from the client to the authenticator but not the other way.

Our model captures the security of the access channels between clients and authenticators. The particular implementation of CTAP2 operates as follows. In the binding phase, the authenticator privately sends its associated secret called pinToken (generated upon powerup) to the trusted client and the pinToken is then stored on the client as the binding state. Later, in the access channel phase, that binding state is used by the bound client to authenticate messages sent to the authenticator. We note that, by the CTAP2 design, each authenticator is associated with a *single* pinToken per power-up, so multiple clients establish multiple access channels with the same authenticator using the *same* pinToken. This limits the security of CTAP2 access channels: for a particular channel from a client to an authenticator to be secure (i.e., no attacker can forge messages sent over that channel), *none* of the clients bound to the same authenticator during the same power-up can be compromised.

Motivated by the above discussion, we distinguish between unforgeability (UF) and strong unforgeability (SUF) for PACA protocols. The former corresponds to the weak level of security discussed above. The latter, captures *strong* fine-grained security where the attacker can compromise any clients except those involved in the access channels for which we claim security. As we explain later (Section 6), SUF also covers certain *forward secrecy* guarantees for authentication. For both notions, we consider a powerful attacker that can manipulate the communication between parties, compromise clients (that are not bound to the target authenticator) to reveal the binding states, and corrupt users (that did not set up the target authenticator) to learn their secret PINs.

Even with the stronger trust assumption (made in UF) on the bound clients, we are unable to prove that CTAP2 realizes the expected security model: we describe an attack that exploits the fact that CTAP2 uses unauthenticated Diffie-Hellman. Since it is important to understand the limits of the protocol, we consider a further refinement of the security models which makes stronger trust assumptions on the binding phase of the protocol. Specifically, in the *trusted binding* setting the

attacker cannot launch active attacks against the client during the binding phase, but it may try to do so against the authenticator, i.e., it cannot launch man-in-the-middle (MITM) attacks but it may try to impersonate the client to the authenticator. We write UF-t and SUF-t for the security levels which consider trusted binding and the distinct security goals outlined above. In summary we propose four notions: by definition SUF is the strongest security notion and UF-t is the weakest one. Interestingly, UF and SUF-t are *incomparable* as established by our separation result discussed in Section 7 and Section 8. Based on our security model, we prove that CTAP2 achieves the weakest UF-t security and show that it is not secure regarding the three stronger notions. Finally, we identify a series of design flaws of CTAP2 and provide suggestions for improvement.

Improving CTAP2 security. CTAP2 cannot achieve UF security because in the binding phase it uses unauthenticated Diffie-Hellman key exchange which is vulnerable to MITM attacks. This observation suggests a change to the protocol which leads to stronger security. Specifically, we propose a generic sPACA protocol (for strong PACA), which replaces the use of unauthenticated Diffie-Hellman in the binding phase with a *password-authenticated key exchange (PAKE)* protocol. Recall that PAKE takes as input a common password and outputs the same random session key for both parties. The key observation is that the client and the authenticator share a value (derived from the user PIN) which can be viewed as a password. By running PAKE with this password as input, the client and the authenticator obtain a strong key which can be used as the binding state to build the access channel. Since each execution of the PAKE (with different clients) results in a fresh independent key, we can prove that sPACA is a SUF-secure PACA protocol. Furthermore, we compare the performance of CTAP2 and sPACA (with proper PAKE instantiations). The results show that our sPACA protocol is also more efficient, so it should be considered for adoption.

Composed security of CTAP2 and WebAuthn. Finally, towards our main goal of the analysis of full FIDO2 (by full FIDO2 we mean the envisioned usage of the two protocols), we study the composition of PlA and PACA protocols (cf. Section 9). The composed protocol, which we simply call PlA+PACA, is defined naturally for an authenticator, user, client, and server. The composition, and the intuition that underlies its security, is as follows. Using PACA, the user (via a client) sets a PIN for the authenticator. This means that only clients that obtain the PIN from the user can "bind" to the authenticator and issue commands that it will accept. In other words, PACA establishes the access channel from the bound client to the authenticator. Then, the challenge-response protocols of PlA run between the server and the authenticator, via a PACA-bound client. The server-side guarantees of PlA are preserved, but now the authenticator can control client access to its credentials using PACA; this composition result is intuitive and easy to prove given our modular formalization.

Interestingly, we formalize an even stronger property that shows that FIDO2 gives end-to-end mutual authentication guarantees between the server and the authenticator when clients and servers are connected by an authenticated server-to-client channel (e.g., a TLS connection). The mutual authentication guarantees extend the PlA guarantees: authenticator, client, and server must all be using the same registration context for authentication to succeed. We note that Transport Layer Security (TLS) provides a server-to-client authenticated channel, and hence this guarantee applies to the typical usage of FIDO2 over TLS. Our results apply to WebAuthn+CTAP2 (under a UF-t adversarial model) and WebAuthn+sPACA (under a SUF adversarial model).

We conclude with an analysis of the role of user gestures in FIDO2. We first show that SUF security offered by sPACA allows the user, equipped with an authenticator that can display a simple session identifier, to detect and prevent attacks from malware that may compromise the states of PACA clients previously bound to the authenticator. (This is not possible for the current version of CTAP2.) We also show how simple gestures can allow a human user to keep track of which server identity is being used in PlA sessions.

Summary. Our analyses clarify the security guarantees FIDO2 should provide for the various parties involved in the most common usage scenario where: 1) the user owns a simple hardware token that is capable of accepting push-button gestures and, optionally, to display a session identifier code (akin to bluetooth pairing codes); 2) the user configures the token with a PIN using a trusted machine; 3) the

user connects/disconnects the token on multiple machines, some trusted, some untrusted, and uses it to authenticated to multiple servers.

In all these interactions, the server is assured that during authentication it can recognize if the same token was used to register a key, and that this token was bound to the client it is talking to since the last power-up (this implies entering the correct PIN *recently*). This guarantee assumes that the client is not corrupted (i.e., the browser window where the user entered the PIN is isolated from malicious code and can run the CTAP2 protocol correctly) and that an active attack against the client via the CTAP2 API to guess the user entered PIN is detected (we know this is the case on the token side, as CTAP2 defines a blocking countermeasure).

Assuming a server-to-client authenticated channel, the user is assured that while it is in possession of the PIN, no one can authenticate on her behalf, except if she provides the PIN to a corrupted browser window. Moreover, the scope of this possible attack is limited to the current power-up period. If we assume that registration was conducted via an honest client, then we know that all authentication sessions with honest clients are placed to the correct server. Finally, if the token is stolen, the attacker still needs to guess the PIN (without locking the authenticator) in order to impersonate the user.

With our proposed modifications, FIDO2 will meet this level of security. Without them, these guarantees will only hold assuming weaker client corruption capabilities and more importantly, the attacker cannot perform active man-in-the-middle attacks during all binding sessions, which may be unrealistic.

2 Preliminaries

Notations. In this paper, $\{0,1\}^*$ denotes the set of all finite-length binary strings (including the empty string ε) and $\{0,1\}^n$ denotes the set of n-bit binary strings. We use $y \leftarrow x$ for assigning a value to a variable. In particular, if f is a deterministic function, $y \leftarrow f(x)$ denotes y taking the output of f on input x; if F is a probabilistic algorithm, we use $y \xleftarrow{\$} F(x)$ for running F on x using fresh random coins and assigning the output to y. We use the wildcard $\cdot$ to indicate any valid input of a function or algorithm.

In Appendix A, we recall the definitions of pseudorandom functions (PRFs), collision-resistant hash function families, message authentication codes (MACs), signature schemes, the computational Diffie-Hellman (CDH) problem and strong CDH (sCDH) problem, as well as the corresponding advantage measures $\mathbf{Adv}^{\mathsf{prf}}$, $\mathbf{Adv}^{\mathsf{coll}}$, $\mathbf{Adv}^{\mathsf{euf\text{-}cma}}$, $\mathbf{Adv}^{\mathsf{euf\text{-}cma}}$, $\mathbf{Adv}^{\mathsf{cdh}}$, $\mathbf{Adv}^{\mathsf{scdh}}$. In Appendix B, we recall the syntax for PAKE and its security of perfect forward secrecy and explicit authentication.

3 Execution Model

The protocols we consider involve four disjoint sets of parties. Formally, the set of parties $\mathcal{P}$ is partitioned into four disjoint sets of users $\mathcal{U}$, authenticators (or tokens for short) $\mathcal{T}$, clients $\mathcal{C}$, and servers $\mathcal{S}$. Each party has a well-defined and non-ambiguous identifier, which one can think of as being represented as an integer; we typically use P, U, T, C, S for identifiers bound to a party in a security experiment and id for the case where an identifier is provided as an input in the protocol syntax. For simplicity, we do not consider certificates or certificate checks but assume any public key associated with a party is supported by a *public key infrastructure (PKI)* and hence certified and bound to the party's identity. This issue arises explicitly only for attestation public keys bound to authenticators in Section 4.

The possible communication channels are represented as double-headed arrows in Figure 2. In FIDO2, the client is a browser and the user-client channel is the browser window, which keeps no long-term state. The authenticator is a hardware token or mobile phone that is connected to the browser via an untrusted link that includes the operating system, some authenticator-specific middleware, and a physical communication channel that connects the authenticator to the machine hosting the

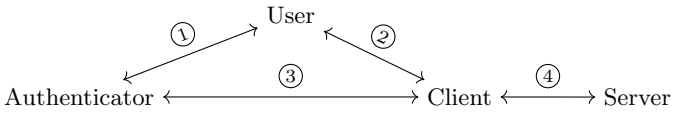

Figure 2: Communication channels

browser. The authenticator exposes a simple interface to the user that allows it to perform a "gesture", confirming some action; ideally the authenticator should also be able to display information to the user (this is natural when using a mobile phone as an authenticator but not so common in USB tokens or smartcards). Following the intuitive definitions of *human-compatible communications* by Boldyreva *et al.* [14], we require that messages sent to the user be *human-readable* and those sent by the user be *human-writable*.[5] The user PIN needs to be *human-memorizable*.

We assume authenticators have a good source of random bits and keep volatile and static (or long-term) storage. Volatile storage is erased every time the device goes through a powerdown/powerup cycle, which we call a *reboot*. Static storage is assumed to be initialized using a procedure carried out under special setup trust assumptions; in the case of this paper we will consider the setup procedures to generate an attestation key pair for the authenticator and to configure a user PIN, i.e., "embedding" the PIN in the authenticator.

Trust model. For each of the protocols we analyze in the paper we specify a trust model, which justifies our proposed security models. Here we state the trust assumptions that are always made throughout the paper. First, human communications (① ②) are authenticated and private. This in practice captures the direct human-machine interaction between the human user and the authenticator device or the client terminal, which involves physical senses and contact that we assume cannot be eavesdropped or interrupted by an attacker. Second, client-authenticator communications (③) are not protected, i.e., neither authenticated nor private. Finally, authenticators are assumed to be tamper-proof, so our models will not consider corruption of their internal state.

Modeling users and their gestures. We do not include in our protocol syntaxes and security models explicit state keeping and message passing for human users, i.e., there are no *session oracles* for users in the security experiments. We shortly explain why this is the case. The role of the user in these protocols is to a) first check that the client is operating on correct inputs, e.g., by looking at the browser window and checking the correct server identity is being used; b) possibly (if the token has the capability to display information) check that the token is operating on inputs consistent to those of the client; and c) finally confirm to the token that this is the case. Therefore, the user itself plays the role of an out-of-band secure channel via which the consistency of information exchanged between the client and the token can be validated.

We model this with a public gesture predicate G that captures the semantics of the user's decision. Intuitively, the user decision $d \in \{0,1\}$ is given by $d = \mathsf{G}(x,y)$, where x and y respectively represent the information conveyed to the user by the client in step a) and by the token in step b) above. Note that x, y may not be input by the user. Tokens with different user interface capabilities give rise to different classes of gesture predicates. For example, if a user can observe a server domain name id on the token display before pressing a button, then we can define the gesture of checking that the token displayed an identifier id that matches the one displayed by the client id^* as $\mathsf{G}(\mathsf{id}^*, \mathsf{id}) = (\mathsf{id}^* \stackrel{?}{=} \mathsf{id})$.

User actions are hardwired into the security experiments as direct inputs to either a client or a token, which is justified by our assumption that users interact with these entities via fully secure channels. We stress that here G is a modeling tool, which captures the sequence of interactions a), b), c) above. Providing a gesture means physical possession of the token, so an attacker controlling only some part of the client machine (e.g., malware) is *not* able to provide a gesture. Moreover, requiring a

[5] We regard understandable information displayed on a machine as human-readable and typing in a PIN or rebooting an authenticator as human-writable.

gesture from the user implies that the user can detect when some action is requested from the token.

4 Passwordless Authentication

We start our analysis with the simpler FIDO2 component protocol, WebAuthn. In order to analyze the authentication security of WebAuthn we first define the syntax and security model for *passwordless authentication (PlA)* protocols.

4.1 Protocol Syntax

A PlA protocol is an interactive protocol among three parties: an authenticator (representing a user), a client, and a server. The authenticator is associated with an attestation public key that is pre-registered to the server. The protocol defines two types of interactions: registration and authentication. In registration the server requests the authenticator to register some initial authentication parameters. If this succeeds, the server can later recognize the same authenticator using a challenge-response protocol.

The possible communication channels are as shown in Figure 2, but we do not include the user. Servers are accessible to clients via a communication channel that models Internet communications.

The state of token T, denoted by st_T, is partitioned into the following static components: i) attestation key pair $(\mathsf{st}_T.ak, \mathsf{st}_T.vk)$; ii) a set of registration contexts $\mathsf{st}_T.\mathsf{rct}$. A server S also has its registration contexts $\mathsf{st}_S.\mathsf{rcs}$. Clients do not keep long-term states. All states are initialized to the empty string ε.

A PlA protocol consists of the following subprotocols:

Key Generation: This algorithm, denoted by Kg, is executed *at most once* for each token; it generates an attestation key pair (ak, vk).

Register: This subprotocol is executed among a token, a client, and a server. The server inputs its identity id_S (e.g., a server domain name) and a set of attestation public keys; the client inputs an intended server identity $\hat{\mathsf{id}}_S$; and the token inputs its static storage. At the end of the subprotocol, if successful, the token and the server obtain new registration contexts, which may be different. Note that the token may successfully complete the protocol while the server may fail to, in the same run. We say the server accepts if it successfully completes and rejects otherwise.

Authenticate: This subprotocol is executed between a token, a client, and a server. The server inputs its identity id_S and registration contexts; the client inputs an intended server identity $\bar{\mathsf{id}}_S$; and the token inputs its registration contexts. At the end of the subprotocol, the server *accepts* or *rejects*. In either case, the registration contexts of the token and the server may be updated.

For both Register and Authenticate, we focus on 2-pass challenge-response protocols with the following structure:

- Server-side computation is split into four procedures: $\mathsf{rchallenge}$ and rcheck for registration, $\mathsf{achallenge}$ and acheck for authentication. The challenge algorithms are probabilistic, which take the server's input to the Register or Authenticate protocol and return a challenge. The check algorithms get the same inputs, the challenge, and a response. rcheck outputs the updated registration contexts rcs that are later input by acheck; acheck outputs a bit b (1 for accept and 0 for reject).

- Client-side computation is modeled as two deterministic functions $\mathsf{rcommand}$ and $\mathsf{acommand}$ that capture possible checks and translations performed by the client before sending the challenges to the token. These algorithms output commands denoted by M_r, M_a respectively, which they generate from the challenges and the server identity. The client may append some information

about the challenge to the token's response before sending it to the server, which is an easy step that we do not model explicitly.

- Token-side computation is modeled as two probabilistic algorithms $\mathsf{rresponse}$ and $\mathsf{aresponse}$ that, on input a command M and the token's input to the corresponding subprotocol, then output a response. $\mathsf{rresponse}$ also outputs the updated registration contexts rct that are later input by $\mathsf{aresponse}$.

Correctness. Correctness imposes that for any server identities $\mathsf{id}_S, \hat{\mathsf{id}}_S, \bar{\mathsf{id}}_S$ the following probability is 1:

$$\Pr\left[\; b = ((\mathsf{id}_S \stackrel{?}{=} \hat{\mathsf{id}}_S) \wedge (\mathsf{id}_S \stackrel{?}{=} \bar{\mathsf{id}}_S)) \;\middle|\; \begin{array}{l} (ak, vk) \stackrel{\$}{\leftarrow} \mathsf{Kg}(\,) \\ c \stackrel{\$}{\leftarrow} \mathsf{rchallenge}(\mathsf{id}_S, vk) \\ M_r \stackrel{\$}{\leftarrow} \mathsf{rcommand}(\hat{\mathsf{id}}_S, c) \\ (R_r, \mathsf{rct}) \stackrel{\$}{\leftarrow} \mathsf{rresponse}(ak, M_r) \\ \mathsf{rcs} \leftarrow \mathsf{rcheck}(\mathsf{id}_S, vk, c, R_r) \\ c' \stackrel{\$}{\leftarrow} \mathsf{achallenge}(\mathsf{id}_S, \mathsf{rcs}) \\ M_a \stackrel{\$}{\leftarrow} \mathsf{acommand}(\bar{\mathsf{id}}_S, c') \\ R_a \stackrel{\$}{\leftarrow} \mathsf{aresponse}(\mathsf{rct}, M_a) \\ b \leftarrow \mathsf{acheck}(\mathsf{id}_S, \mathsf{rcs}, c', R_a) \end{array} \right]$$

Intuitively, correctness requires that the server always accepts an authentication that is consistent with a prior registration, and only the intended server should perform the registration and authentication. Note that the latter helps prevent the server-in-the-middle attack identified in [33].

4.2 Security Model

Trust model. Before defining security we clarify that there are no security assumptions on the communication channels shown in Figure 2. Again, authenticators are assumed to be tamper-proof, so the model will not consider corruption of their internal state. We assume the key generation stage, where the attestation key pair is created and installed in the authenticator, is either carried out within the token itself, or it is performed in a trusted context that leaks nothing about the attestation secret key.

Session oracles. Each party $P \in \mathcal{T} \cup \mathcal{S}$ is associated with a set of session oracles $\pi_P^{i,j}$, where we need to manage two types of sessions corresponding to registration and authentication. We omit session oracles for clients, since all they do can be performed by the adversary. For servers and tokens, session oracles are structured as follows: $\pi_P^{i,0}$ refers to a pending or completed registration session, whereas $\pi_P^{i,j}$ for $j \geq 1$ refers to the j-th pending or completed authentication session associated with $\pi_P^{i,0}$ after this registration completed. A party's static storage is shared among all its session oracles. The security experiment also manages the attestation public keys of different tokens and provides them to the server oracles as needed.

Security experiment. The security experiment is run between a challenger and an adversary $\mathcal{A}$. At the beginning of the experiment, the challenger runs $(ak_T, vk_T) \stackrel{\$}{\leftarrow} \mathsf{Kg}(\,)$ for all $T \in \mathcal{T}$ to generate their attestation key pairs. The adversary $\mathcal{A}$ is given all attestation public keys and is allowed to interact with session oracles via the following queries:

• $\mathsf{Start}(\pi_S^{i,j}, \mathsf{id}_S, T)$. The challenger instructs $\pi_S^{i,j}$ to execute $\mathsf{rchallenge}$ (if $j = 0$) or $\mathsf{achallenge}$ (if $j > 0$) to start the registration (for the given token) or authentication (for the $\pi_S^{i,0}$ registration contexts) for the indicated server identity, and generate a challenge c, which is given to the adversary.
• $\mathsf{Challenge}(\pi_T^{i,j}, M)$. The challenger delivers command M to $\pi_T^{i,j}$, which processes the command using $\mathsf{rresponse}$ (if $j = 0$) or $\mathsf{aresponse}$ (if $j > 0$) and return the result R to the adversary.
• $\mathsf{Complete}(\pi_S^{i,j}, R)$. This query delivers an authenticator response to a server oracle, which processes the response using rcheck (if $j = 0$) or acheck (if $j > 0$) and return the result to the adversary.

We assume without loss of generality that each query is only called once for each instance and allow the adversary to get the full state of the server via Start and Complete queries.

Partners. A server registration oracle $\pi_S^{i,0}$ and a token registration oracle $\pi_T^{k,0}$ are partnered if their views are consistent. We take these views as a *session identifier* sid that must be defined by the protocol and must be the same for both parties, usually as a function of the communication trace. A server authentication oracle $\pi_S^{i,j}$ for $j > 0$ and a token authentication oracle $\pi_T^{k,l}$ for $k > 0$ are partnered if: i) they agree on sid; and ii) $\pi_S^{i,0}$ and $\pi_T^{k,0}$ are partnered.

We note that a crucial aspect of this definition is that the authentication session partnership only holds if the token and the server are also partnered for the associated registration sessions: a credential registered in a server should not be used to authenticate a token using another credential.

Advantage measure. Let Π be a PlA protocol. We define the passwordless authentication advantage $\mathbf{Adv}_{\Pi}^{\mathsf{pla}}(\mathcal{A})$ as the probability that a server oracle accepts but it is not uniquely partnered with a token oracle. In other words, a secure PlA protocol guarantees that, if a server session accepts, then there exists a unique token session which has derived the same session identifier, and no other server session has derived the same session identifier.

5 The W3C Web Authentication Protocol

In this section, we present the present the cryptographic core of W3C's Web Authentication (WebAuthn) protocol [15] of FIDO2 and analyze its seurity.

Protocol Description. We show the core cryptographic operations of WebAuthn in Figure 3 in accordance with PlA syntax.[6] For WebAuthn, a server identity is an effective domain (e.g., a hostname) of the server URL. The attestation key pair is generated by the key generation algorithm Kg of a signature scheme $\mathsf{Sig} = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Ver})$. (Note that WebAuthn supports the RSASSA-PKCS1-v1_5 and RSASSA-PSS signature schemes [29].) In Figure 3, we use H to denote the SHA-256 hash function and λ to denote the default parameter 128 (in order to accommodate potential parameter changes). WebAuthn supports two types of operations: Registeration and Authentication (cf. Figure 1 and Figure 2 in [15]), respectively corresponding to the PlA Register and Authenticate subprotocols. In the following description, we assume each token is registered at most once for a server; this is without loss of generality since otherwise one can treat the one token as several tokens sharing the same attestation key pair.

- In registration, the server generates a random string c of length at least $\lambda = 128$ bits and a random 512-bit user id *uid*, then sends it to the client with its identity id_S. Then, the client checks if the received id_S matches its input (i.e., the intended server), and passes the challenge (where c is hashed) to the token. The token generates a key pair with Sig.Kg, sets the signature counter n to 0,[7] and samples a credential id *cid* of length at least $\lambda = 128$ bits, then generates an attestation signature and sends the credential and signature to the client as a response; the token also inserts the generated credential into its registration contexts. Upon receiving the response, the server checks the validity of the attestation signature and inserts the credential into its registration contexts.

- In authentication, the server also generates a random string c but no *uid* is sampled, then sends it to the client with its identity id_S. Then, the client checks if the received id_S matches its input and passes the challenge (where c is hashed) to the token. The token retrieves the credential associated with the authenticating server id_S from its registration contexts, then it increments the signature counter, computes a signature of the challenge, and sends them to the client together with the retrieved credential id and user id. Upon receiving the response, the server checks the validity of

[6] We do not include the WebAuthn explicit reference to user interaction/gestures at this point, as this will be later handled by our PACA protocol.

[7] The signature counter is mainly used to detect clone tokens, but it also helps in preventing replay attacks.

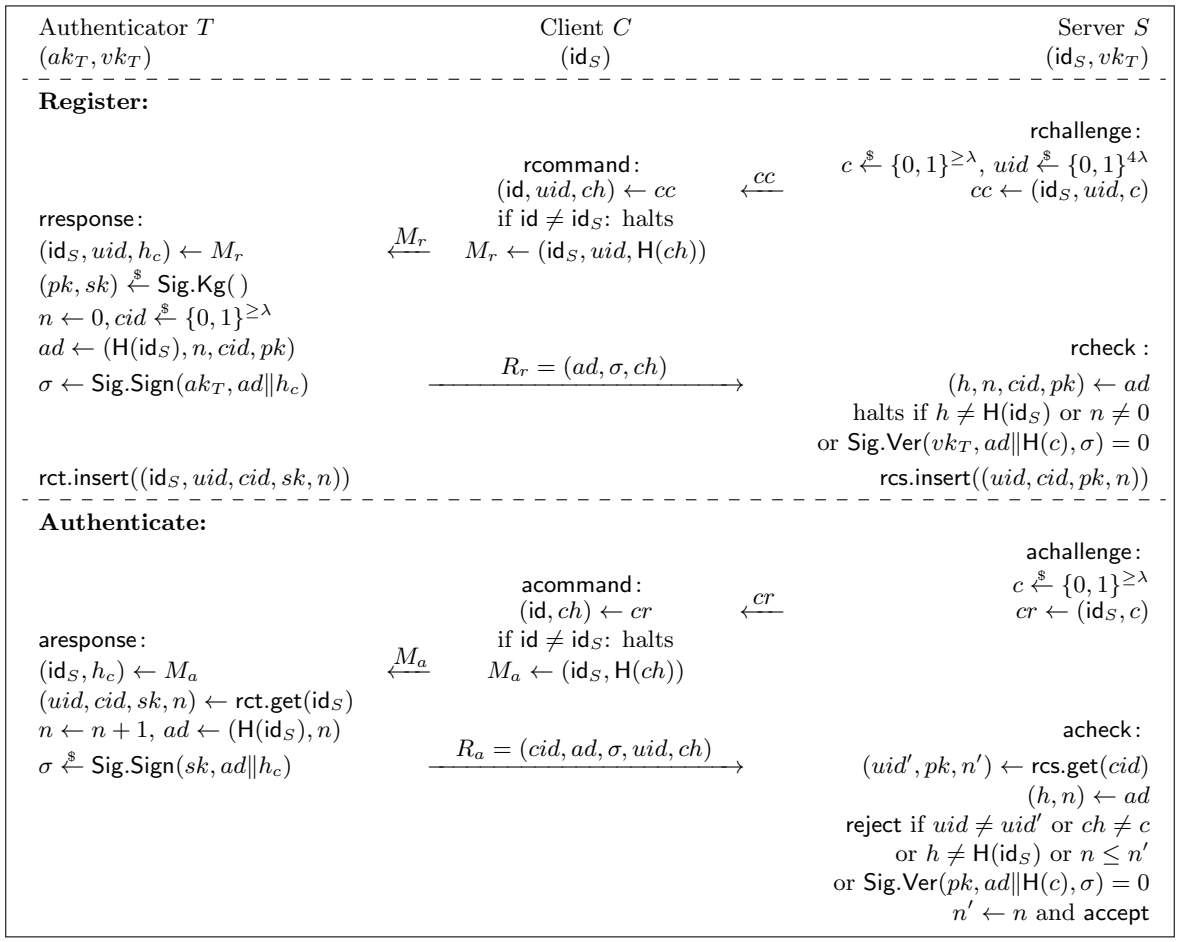

Figure 3: The WebAuthn protocol

the signature and the signature counter; it updates the signature counter and accepts if all checks pass.

It is straightforward to check that WebAuthn is a correct PlA protocol.

WebAuthn Analysis. The following theorem assesses PlA security of WebAuthn, which is proved in Appendix D.1 using $(ad, H(c))$ as the session identifier.

Theorem 1 *For any efficient adversary $\mathcal{A}$ that makes at most q_S queries to* Start *and q_C queries to* Challenge*, there exist efficient adversaries $\mathcal{B}, \mathcal{C}$ such that (recall $\lambda = 128$):*

$$\mathbf{Adv}^{\mathsf{pla}}_{\mathrm{WebAuthn}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{coll}}_{\mathsf{H}}(\mathcal{B}) + q_\mathsf{S}\mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{Sig}}(\mathcal{C}) + (q_\mathsf{S}^2 + {q_\mathsf{C}}^2) \cdot 2^{-\lambda}.$$

The security guarantees for the WebAuthn instantiations follow from the results proving RSASSA-PKCS1-v1_5 and RSASSA-PSS to be EUF-CMA in the random oracle model under the RSA assumption [13, 27] and the assumption that SHA-256 is collision-resistant.

6 PIN-Based Access Control for Authenticators

In this section, we define the syntax and security model for *PIN-based access control for authenticators* (PACA) protocols. The goal of the protocol is to ensure that after PIN setup and possibly an arbitrary

number of authenticator reboots, the user can employ the client to issue PIN-authorized commands to the token, which the token can use for access control, e.g., to unlock built-in functionalities that answer client commands.

6.1 Protocol Syntax

A PACA protocol is an interactive protocol involving a human user, an authenticator (or token for short), and a client. The state of token T, denoted by st_T, consists of static storage $\mathsf{st}_T.\mathsf{ss}$ that remains intact across reboots and volatile storage $\mathsf{st}_T.\mathsf{vs}$ that gets reset after each reboot. $\mathsf{st}_T.\mathsf{ss}$ is comprised of a private secret $\mathsf{st}_T.\mathsf{s}$ and a public retries counter $\mathsf{st}_T.\mathsf{n}$, where the latter is used to limit the maximum number of consecutive failed active attacks (e.g., PIN guessing attempts) against the token. $\mathsf{st}_T.\mathsf{vs}$ is partitioned into powerup state $\mathsf{st}_T.\mathsf{ps}$ and binding states $\mathsf{st}_T.\mathsf{bs}_i$ (together denoted by $\mathsf{st}_T.\mathsf{bs}$). A client C may also have binding states, denoted by $\mathsf{bs}_{C,j}$. All states are initialized to the empty string ε.

A PACA protocol is associated with an arbitrary public gesture predicate G and consists of the following algorithms and subprotocols, all of which can be executed a number of times, except if stated otherwise:

Reboot: This algorithm represents a powerdown/powerup cycle and it is executed by the authenticator *with mandatory user interaction*. We use $\mathsf{st}_T.\mathsf{vs} \xleftarrow{\$} \mathsf{reboot}(\mathsf{st}_T.\mathsf{ss})$ to denote the execution of this algorithm, which inputs its static storage and resets all volatile storage. Note that one should always run this algorithm to power up the token at the beginning of PACA execution.

Setup: This subprotocol is executed *at most once* for each authenticator. The user inputs a PIN through the client and the token inputs its volatile storage. In the end, the token sets up its static storage and the client (and through it the user) gets an indication of whether the protocol completed successfully.

Bind: This subprotocol is executed by the three parties to establish an access channel over which commands can be issued. The user inputs its PIN through the client, whereas the token inputs its static storage and powerup state. At the end of this phase, in the case of success, the token and the client each get a binding state; otherwise, no binding states are set. However, in either case, the token may update its static retries counter.[8] We assume the client always initiates this subprotocol once it gets the PIN from the user.

Authorize: This algorithm allows a client to generate authorized commands for the token. The client inputs a binding state $\mathsf{bs}_{C,j}$ and a command M. We denote $(M,t) \xleftarrow{\$} \mathsf{authorize}(\mathsf{bs}_{C,j}, M)$ as the generation of an authorized command.

Validate: This algorithm allows a token to verify authorized commands sent by a client with respect to a user decision. The token inputs a binding state $\mathsf{st}_T.\mathsf{bs}_i$, an authorized command (M,t), and a user decision $d = \mathsf{G}(x,y)$. We denote $b \leftarrow \mathsf{validate}(\mathsf{st}_T.\mathsf{bs}_i, (M,t), d)$ as the validation performed by the token to obtain an accept or reject indication.

Correctness. For an arbitrary public predicate G, we consider any token T and any sequence of PACA subprotocol executions that includes the following (which may not be consecutive): i) a Reboot of T; ii) a successful Setup using PIN fixing $\mathsf{st}_T.\mathsf{ss}$ via some client; iii) a Bind with PIN creating token-side binding state $\mathsf{st}_T.\mathsf{bs}_i$ and client-side binding state $\mathsf{bs}_{C,j}$ at a client C; iv) authorization of command M by C as $(M,t) \xleftarrow{\$} \mathsf{authorize}(\mathsf{bs}_{C,j}, M)$; and v) validation by T as $b \leftarrow \mathsf{validate}(\mathsf{st}_T.\mathsf{bs}_i, (M,t), d)$. If no Reboot of T is executed after iii), then correctness requires that $b = 1$ if and only if $\mathsf{G}(x,y) = 1$ (i.e., $d = 1$) holds.

[8] When such an update is possible, and to avoid concurrency issues, it is natural to assume that the token excludes concurrent Bind executions.

REMARK. The above PACA syntax may seem overly complex but it is actually difficult (if not impossible) to decompose. First, Setup and Bind share the same powerup state generated by Reboot so cannot be separated into two independent procedures. Then, although Authorize and Validate together can independently model an access channel, detaching them from PACA makes it difficult to define security in a general way: Bind may not establish random symmetric keys; it could, for instance, output asymmetric key pairs.

6.2 Security Model

Trust model. Before defining our security model, we first state the assumed security properties for the involved communication channels, as shown in Figure 2 excluding the client-server channel. We assume that Setup is carried out over an authenticated channel where the adversary can only eavesdrop communications between the client and authenticator; this is a necessary assumption, as there are no pre-established authentication parameters between the parties.

Session oracles. To capture multiple sequential and parallel Bind executions, each party $P \in \mathcal{T} \cup \mathcal{C}$ is associated with a set of session oracles $\pi_P^1, \pi_P^2, \ldots$, where π_P^i models the i-th protocol instance of P. For clients, session oracles are totally independent from each other and they are assumed to be available throughout the protocol execution. For tokens, the static storage and powerup states are maintained by the security experiment and shared by all oracles of the same token; session oracles capture only binding states (if any) that are reset, i.e., becoming *invalid*, after rebooting. This means the adversary is thereafter blocked access to such oracles. However, these oracles are kept in the experiment to capture strong authentication guarantees across reboots as described later.

Security experiment. The security experiment is executed between a challenger and an adversary $\mathcal{A}$. At the beginning of the experiment, the challenger fixes an arbitrary distribution $\mathscr{D}$ over a PIN dictionary $\mathcal{PIN}$ associated with PACA; it then samples independent user PINs according to $\mathscr{D}$, denoted by $\langle \mathsf{pin}_U \xleftarrow{\mathscr{D}} \mathcal{PIN} \rangle_{U \in \mathcal{U}}$. Without loss of generality, we assume each user holds only one PIN. The challenger also initializes states of all oracles to the empty string. Then, $\mathcal{A}$ interacts with the challenger via the following queries:

- $\mathsf{Reboot}(T)$. The challenger runs Reboot for token T, marking all previously used instances π_T^i (if any) as *invalid*[9] and setting $\mathsf{st}_T.\mathsf{vs} \xleftarrow{\$} \mathsf{reboot}(\mathsf{st}_T.\mathsf{ss})$.
- $\mathsf{Setup}(\pi_T^i, \pi_C^j, U)$. The challenger inputs pin_U through π_C^j and runs Setup between π_T^i and π_C^j; it returns the trace of communications to $\mathcal{A}$. After this query, T is *set up*, i.e., $\mathsf{st}_T.\mathsf{ss}$ is set and available, for the rest of the experiment. Oracles created in this query, i.e., π_T^i and π_C^j, must never have been used before and are always marked *invalid* after Setup completion.[10]
- $\mathsf{Execute}(\pi_T^i, \pi_C^j)$. The challenger runs Bind between π_T^i and π_C^j using the same pin_U that set up T; it returns the trace of communications to $\mathcal{A}$. This query allows the adversary to access honest Bind executions in which it can only take passive actions, i.e., eavesdropping. The resulting binding states on both sides are kept as $\mathsf{st}_T.\mathsf{bs}_i$ and $\mathsf{bs}_{C,j}$ respectively.
- $\mathsf{Connect}(T, \pi_C^j)$. The challenger asks π_C^j to initiate the Bind subprotocol with T using the same pin_U that set up T; it returns the first message sent by π_C^j to $\mathcal{A}$. Note that no client oracles can be created for active attacks if Connect queries are disallowed, since we assume the client is the initiator of Bind. This query allows the adversary to launch an active attack against a client oracle.
- $\mathsf{Send}(\pi_P^i, m)$. The challenger delivers m to π_P^i and returns its response (if any) to $\mathcal{A}$. If π_P^i completes the Bind subprotocol, then the binding state is kept as $\mathsf{st}_T.\mathsf{bs}_i$ for a token oracle and as $\mathsf{bs}_{C,i}$ for a client oracle. This query allows the adversary to launch an active attack against a token oracle or completing an active attack against a client oracle.
- $\mathsf{Authorize}(\pi_C^j, M)$. The challenger asks π_C^j to authorize command M; it returns the authorized command $(M, t) \xleftarrow{\$} \mathsf{authorize}(\mathsf{bs}_{C,j}, M)$.

[9] All queries are ignored if they refer to an oracle π_P^i marked as invalid.

[10] Session oracles used for Setup are separated since they may cause ambiguity in defining session identifiers for Bind sessions.

• $\mathsf{Validate}(\pi_T^i, (M,t))$. The challenger asks π_T^i (that received a user decision d) to validate (M,t); it returns the validation result $b \leftarrow \mathsf{validate}(\mathsf{st}_T.\mathsf{bs}_i, (M,t), d)$.
• $\mathsf{Compromise}(\pi_C^j)$. The challenger returns $\mathsf{bs}_{C,j}$ and marks π_C^i as *compromised*.
• $\mathsf{Corrupt}(U)$. The challenger returns pin_U and marks pin_U as *corrupted*.

Partners. We say a token oracle π_T^i and a client oracle π_C^j are each other's *partner* if they have both completed their Bind executions and their views are consistent. We take these views as a *session identifier* sid that must be defined by the protocol and must be the same for both parties. Then, we also say π_C^j is T's partner (and hence T may have more than one partners). Note that, as mentioned before, if a token is rebooted then all of its existing session oracles (if any) are invalidated. A *valid* partner refers to a valid session oracle.

Security goals. We define four levels of security for a PACA protocol Π. All advantage measures define PAKE-like security: the adversary's winning probability should be negligibly larger than that of the trivial attack of guessing the user PIN (known as an *online dictionary attack* with more details in Appendix B).

Unforgeability (UF). We define $\mathbf{Adv}_\Pi^{\mathsf{uf}}(\mathcal{A})$ as the probability that there exists a token oracle π_T^i that accepts an authorized command (M,t) for gesture G and at least one of the following conditions does *not* hold:

1) G approves M, i.e., $\mathsf{G}(x,y) = 1$;

2) (M,t) was output by one of T's valid partners π_C^j.

The adversary must be able to trigger this event without: i) corrupting pin_U that was used to set up T, before π_T^i accepted (M,t); or ii) compromising any of T's partners created after T's last reboot and before π_T^i accepted (M,t).

The above captures the attacks where the attacker successfully makes a token accept a forged command, without corrupting the user PIN used to set up the token or compromising any of the token's partners. In other words, a UF-secure PACA protocol protects the token from unauthorized access even if it is stolen and possessed by an attacker. Nevertheless, UF considers only weak security for access channels, i.e., compromising one channel could result in compromising all channels (with respect to the same token after its last reboot).

Unforgeability with trusted binding (UF-t). We define $\mathbf{Adv}_\Pi^{\mathsf{uf\text{-}t}}(\mathcal{A})$ the same as $\mathbf{Adv}_\Pi^{\mathsf{uf}}(\mathcal{A})$ except that the adversary is *not* allowed to make $\mathsf{Connect}$ queries.

As mentioned before, the attacker is now forbidden to launch active attacks against clients (that input user PINs) during binding; it can still, however, perform active attacks against tokens. This restriction captures the minimum requirement for proving the security of CTAP2 (using our model), which is the main reason we define UF-t. Clearly, UF security implies UF-t security.

Strong unforgeability (SUF). We define $\mathbf{Adv}_\Pi^{\mathsf{suf}}(\mathcal{A})$ as the UF advantage, with one more condition captured:

3) π_T^i and π_C^j are each other's unique valid partner.

More importantly, the adversary considered in this strong notion is allowed to compromise T's partners, provided that it has not compromised π_C^j. It is also allowed to corrupt pin_U used to set up T even before the command is accepted, *as long as* π_T^i has set its binding state.

The above captures similar attacks considered in UF but in a strong sense, where the attacker is allowed to compromise the token's partners. This means SUF considers strong security for access channels, i.e., compromising any channel does not affect other channels. It hence guarantees a unique binding between an accepted command and an access channel (created by uniquely partnered token and client oracles by running Bind), which explains condition 3). Finally, the attacker is further allowed to corrupt the user PIN *immediately* after the access channel establishment. This guarantees *forward secrecy* for access channels, i.e., once the channel is created its security will no longer be affected by later PIN corruption. Note that SUF security obviously implies UF security.

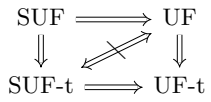

Figure 4: Relations between PACA security notions.

Strong unforgeability with trusted binding (SUF-t). For completeness we can also define $\mathbf{Adv}_{\Pi}^{\mathsf{suf\text{-}t}}(\mathcal{A})$, where the adversary is *not* allowed to make Connect queries. Again, it is easy to see that SUF security implies SUF-t security.

Relations between PACA security notions. Figure 4 shows the implication relations among our four defined notions. Note that UF and SUF-t do not imply each other, for which we will give separation examples in Sections 7 and 8.

Improving (S)UF-t security with user confirmation. Trusted binding excludes active attacks against the client (during binding), but online dictionary attacks are still possible against the token. Such attacks can be mitigated by requiring user confirmation (e.g., pressing a button) for Bind execution, such that only honest Bind executions will be approved when the token is possessed by an honest user. We argue that the confirmation overhead is quite small for CTAP2-like protocols since the user has to type its PIN into the client anyway; the security gain is meaningful as now *no* online dictionary attacks (that introduce non-negligible adversarial advantage) can happen to unstolen tokens.

A practical implication of SUF security. We note that SUF security has a practical meaning: an accepted command can be traced back to a unique access channel. This means that an authenticator that allows a user to confirm a session identifier (that determines the channel) for a command can allow a human user to detect rogue commands issued by an adversary (e.g., malware) that compromised one of the token's partners (e.g., browsers).

PACA security bounds. In our theorems for PACA security shown later, we fix q_{S} (i.e., the number of Setup queries) as one adversarial parameter to bound the adversary's success probability of online dictionary attacks (e.g., the first bound term in Theorem 2 and the PAKE advantage term in Theorem 3), while for PAKE security the number of Send queries q_s is used (see Appendix B or [11] for example). This is because PACA has a token-side retries counter to limit the total number of failed PIN guessing attempts (across reboots).

7 The Client to Authenticator Protocol v2.0

In this section, we present the cryptographic core of the FIDO Alliance's CTAP2, analyze its security using PACA model, and make suggestions for improvement.

Protocol Description. CTAP2's cryptographic core lies in its authenticator API[11] which we show in Figure 5 in accordance with PACA syntax. One can also refer to its specification (Figure 1, [1]) for a command-based description.[12] The PIN dictionary $\mathcal{PIN}$ of CTAP2 consists of 4∼63-byte strings.[13] In Figure 5, the client inputs an arbitrary user PIN $\mathsf{pin}_U \in \mathcal{PIN}$. We use $\mathsf{ECKG}_{\mathbb{G},G}$ to denote the key generation algorithm of the NIST P-256 elliptic-curve Diffie-Hellman (ECDH) [25], which samples an elliptic-curve secret and public key pair (a, aG), where G is an elliptic-curve point that generates a cyclic group $\mathbb{G}$ of prime order $|\mathbb{G}|$ and a is chosen at random from the integer set $\{1, \ldots, |\mathbb{G}| - 1\}$.

[11] The rest of CTAP2 does not focus on security but specifies transport-related behaviors like message encoding and transport-specific bindings.

[12] There the command used for accessing the retries counter $\mathsf{st}_T.\mathsf{n}$ is omitted because PACA models it as public state. Commands for PIN resets are also omitted and left for future work, but capturing those is not hard by extending our analysis since CTAP2 changes PIN by simply running the first part of Bind (to establish the encryption key and verify the old PIN) followed by the last part of Setup (to set a new PIN). Without PIN resets, our analysis still captures CTAP2's core security aspects and our PACA model becomes more succinct.

[13] PINs memorized by users are at least 4 Unicode characters and of length at most 63 bytes in UTF-8 representation.

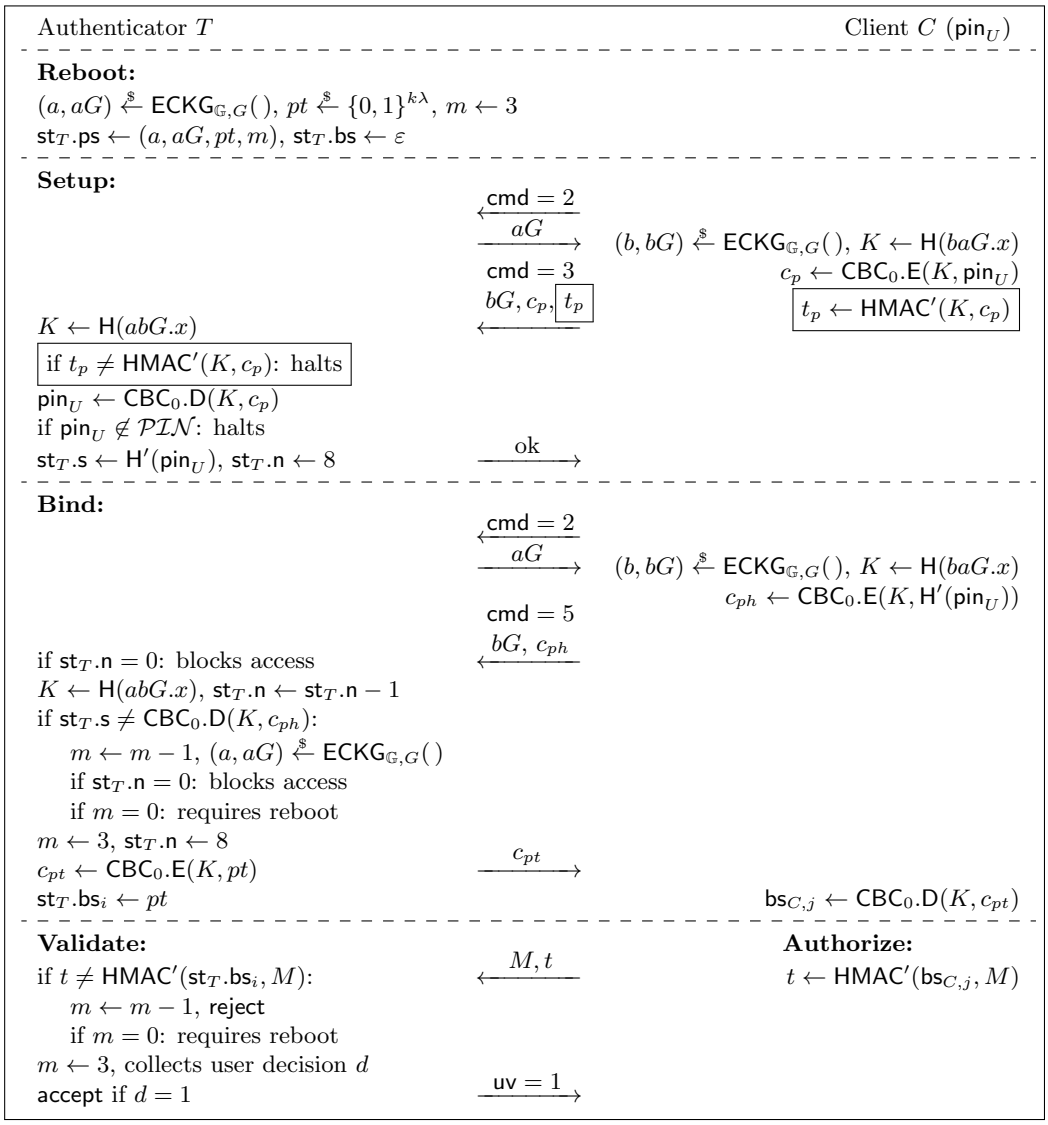

Figure 5: The CTAP2 protocol (and CTAP2* that excludes the boxed contents).

Let H denote the hash function SHA-256 [31] and H' denote SHA-256 with output truncated to the first $\lambda = 128$ bits; $\mathsf{CBC}_0 = (\mathcal{K}, \mathsf{E}, \mathsf{D})$ denotes the (deterministic) encryption scheme AES-256-CBC [20] with fixed IV $= 0$; HMAC' denotes the MAC HMAC-SHA-256 [9] with output truncated to the first $\lambda = 128$ bits. Note that we use the symbol λ to denote the block size in order to accommodate parameter changes in future versions of CTAP2.

- Reboot generates $\mathsf{st}_T.\mathsf{ps}$ by running $\mathsf{ECKG}_{\mathbb{G},G}$, sampling a $k\lambda$-bit pinToken pt (where $k \in \mathbb{N}_+$ can be any fixed parameter, e.g., $k = 2$ for a 256-bit pt), and resetting the mismatch counter $m \leftarrow 3$ that limits the maximum number of consecutive mismatches. It also erases the binding state $\mathsf{st}_T.\mathsf{bs}$ (if any).

- Setup is essentially an unauthenticated ECDH followed by the client transmitting the (encrypted) user PIN to the token. The shared encryption key is derived from hashing the x-coordinate of the ECDH result. A HMAC' tag of the encrypted PIN is also attached for authentication; but as we

will show this is actually useless. The token checks if the tag is correct and if the decrypted PIN pin_U is valid; if so, it sets the static secret $\mathsf{st}_T.\mathsf{s}$ to the PIN hash and sets the retries counter $\mathsf{st}_T.\mathsf{n}$ to the default value 8.

- Bind also involves an unauthenticated ECDH but followed by the transmission of the encrypted PIN hash. The token checks if the decrypted PIN hash matches its stored static secret; if so, it resets the retries counter, sends back the encrypted pinToken, and uses its pinToken as the binding state $\mathsf{st}_T.\mathsf{bs}_i$; the client then uses the decrypted pinToken as its binding state $\mathsf{bs}_{C,j}$. If the check fails, the retries counter $\mathsf{st}_T.\mathsf{n}$ and mismatch counter m are both decremented by 1. If $\mathsf{st}_T.\mathsf{n} = 0$, the token blocks further access unless being reset to factory default state, i.e., erasing all static and volatile state. If $m = 0$, it requires a reboot to enforce user interaction (and hence user detectability).
- Authorize generates an authorized command by attaching a HMAC' tag.
- Validate accepts the command if and only if the tag is correct and the user gesture approves the command. The default CTAP2 gesture predicate G_1 always returns true, since only physical user presence is required. The mismatch counter is also updated to trigger user interaction.

It is straightforward to check that CTAP2 is a correct PACA protocol.

CTAP2 Analysis. The session identifier of CTAP2 is defined as the full communication trace of the Bind execution.

Insecurity of CTAP2. It is not hard to see that CTAP2 is not UF-secure (and hence not SUF-secure). An attacker can query $\mathsf{Connect}$ to initiate the Bind execution of a client oracle that inputs the user PIN, then impersonate the token to get the PIN hash, and finally use it to get the secret binding state pt from the token. CTAP2 is not SUF-t-secure either because compromising any partner of the token reveals the common binding state pt used to access all token oracles.

UF-t security of CTAP2. The following theorem (proved in Appendix D.3) confirms UF-t security of CTAP2, where the hash function H (with fixed 256-bit input) and the truncated HMAC HMAC' are modeled as random oracles $\mathcal{H}_1, \mathcal{H}_2$.

Theorem 2 *Let $\mathscr{D}$ be an arbitrary distribution over $\mathcal{PIN}$ with min-entropy $h_{\mathscr{D}}$. For any efficient adversary $\mathcal{A}$ making at most $q_{\mathsf{S}}, q_{\mathsf{E}}, q_{\mathsf{R}}, q_{\mathsf{V}}$ queries respectively to* $\mathsf{Setup}, \mathsf{Execute}, \mathsf{Reboot}, \mathsf{Validate}$*, and $q_{\mathcal{H}}$ random oracle queries to $\mathcal{H}_2$, there exist efficient adversaries $\mathcal{B}, \mathcal{C}, \mathcal{D}$ such that (recall $\lambda = 128$):*

$$\begin{aligned}\mathbf{Adv}^{\mathsf{uf\text{-}t}}_{\mathrm{CTAP2}}(\mathcal{A}) \le\ & 8q_{\mathsf{S}} \cdot 2^{-h_{\mathscr{D}}} + (q_{\mathsf{S}} + q_{\mathsf{E}})\mathbf{Adv}^{\mathsf{scdh}}_{\mathbb{G},G}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{coll}}_{\mathsf{H}'}(\mathcal{C}) \\ & + 2(q_{\mathsf{S}} + q_{\mathsf{E}})\mathbf{Adv}^{\mathsf{prf}}_{\mathrm{AES}-256}(\mathcal{D}) + q_{\mathsf{V}} \cdot 2^{-k\lambda} + q_{\mathsf{S}} q_{\mathcal{H}} \cdot 2^{-2\lambda} \\ & + (12q_{\mathsf{S}} + 2|\mathcal{U}|q_{\mathsf{R}}q_{\mathsf{E}} + q_{\mathsf{R}}^2 q_{\mathsf{E}} + (k+1)^2 q_{\mathsf{E}} + q_{\mathsf{V}}) \cdot 2^{-\lambda}.\end{aligned}$$

SUF-t $\not\Rightarrow$ UF. Note that we can modify CTAP2 to achieve SUF-t security by using independent pinTokens for each Bind execution, but this is not UF-secure due to unauthenticated ECDH. This shows that SUF-t does not imply UF.

CTAP2 improvement. Here we make suggestions for improving CTAP2 per se, but we advocate the adoption of our proposed efficient PACA protocol with stronger SUF security in Section 8.

Setup simplification. First, we notice that the Setup authentication procedures (boxed in Figure 5) are useless, since there are no pre-established authentication parameters between the token and client. In particular, a MITM attacker can pick its own aG to compute the shared key K and generate the authentication tag. More importantly, CTAP2 uses the same key K for both encryption and authentication, which is considered bad practice and the resulting security guarantee is elusive; this is why we have to model HMAC' as a random oracle. Therefore, we suggest removing those redundant authentication procedures (or using checksums), then the resulting protocol, denoted by CTAP2*, is

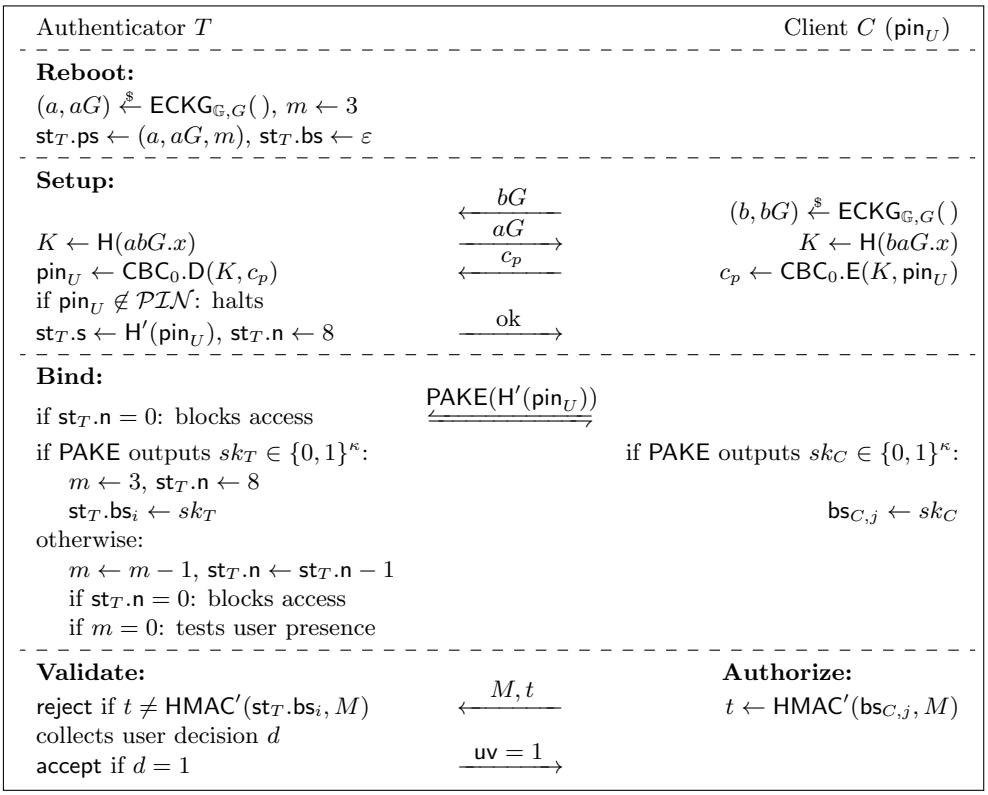

Figure 6: The sPACA protocol

also UF-t-secure, with the proof in Appendix D.4 where HMAC' is treated as an EUF-CMA-secure MAC.[14]

Unnecessary reboots. In order to prevent attacks that block the token without user interaction, CTAP2 requires a token reboot after 3 consecutive failed binding attempts. Such reboots do not enhance security as the stored PIN hash is not updated, but they could cause usability issues since reboots invalidate all established access channels by erasing the existing binding states. We therefore suggest replacing reboots with tests of user presence (e.g., pressing a button) that do not affect existing bindings. Note that reboots are also introduced for user interaction in Validate executions; this however is completely useless when CTAP2 already requires a test of user presence before accepting each command.

User confirmation for binding. As discussed at the end of Section 6, we suggest CTAP2 require user confirmation for Bind executions to improve security.

8 The Secure PACA Protocol

In this section, we propose a generic PACA protocol that we call sPACA for *secure PACA*, prove its SUF security, and compare its performance with CTAP2 when instantiating the underlying PAKE of sPACA with CPace [23].

Protocol Description. We purposely design our sPACA protocol following CTAP2 such that the required modification is minimized if sPACA is adopted. As shown in Figure 6, sPACA employs

[14]Note that HMAC-SHA-256 has been proved to be a PRF (and hence EUF-CMA) assuming SHA-256's compression function is a PRF [8].

Table 1: Performance comparison of CTAP2 and sPACA for binding.

Protocol	Flow	Token			Client			Communication
		exp	hash	AES	exp	hash	AES	($\lambda = 128$)
CTAP2	4	2	1	$2k$	2	2	$2k$	$4\lambda + 2k\lambda$ (e.g., $k = 2$)
sPACA[CPace]	3	2	4	0	2	5	0	$4\lambda + 2\kappa$ (e.g., $\kappa = 256$)

the same PIN dictionary $\mathcal{PIN}$ and cryptographic primitives as CTAP2 and additionally relies on a PAKE protocol PAKE initiated by the client. Compared to CTAP2, sPACA does not have pinTokens, but instead establishes independent random binding states in Bind executions by running PAKE between the token and the client (that inputs the user PIN) on the shared PIN hash; it also excludes unnecessary reboots. We also note that the length of session keys $sk_T, sk_C \in \{0,1\}^\kappa$ established by PAKE is determined by the concrete PAKE instantiation; typically $\kappa \in \{224, 256, 384, 512\}$ when the keys are derived with a SHA-2 hash function.

sPACA Analysis. The session identifier of sPACA is simply that of PAKE.

SUF security of sPACA. The following theorem (proved in Appendix D.5) confirms SUF security of sPACA by modeling H as a random oracle.

Theorem 3 *Let* PAKE *be a 3-pass protocol where the client is the initiator and let* $\mathscr{D}$ *be an arbitrary distribution over* $\mathcal{PIN}$ *with min-entropy* $h_{\mathscr{D}}$*. For any efficient adversary* $\mathcal{A}$ *making at most* $q_\mathsf{S}, q_\mathsf{C}, q_\mathsf{E}$ *queries respectively to* $\mathsf{Setup}, \mathsf{Connect}, \mathsf{Execute}$*, there exist efficient adversaries* $\mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}, \mathcal{F}$ *such that:*

$$\begin{aligned}\mathbf{Adv}^{\mathsf{suf}}_{\mathrm{sPACA}}(\mathcal{A}) \leq\ & q_\mathsf{S}\mathbf{Adv}^{\mathsf{cdh}}_{\mathbb{G},G}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{coll}}_{\mathsf{H}'}(\mathcal{C}) + 2q_\mathsf{S}\mathbf{Adv}^{\mathsf{prf}}_{\mathrm{AES}-256}(\mathcal{D}) \\ & + \mathbf{Adv}_{\mathsf{PAKE}}(\mathcal{E}, 16q_\mathsf{S} + 2q_\mathsf{C}, h_{\mathscr{D}}) + (q_\mathsf{C} + q_\mathsf{E})\mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{HMAC}'}(\mathcal{F}) + 12q_\mathsf{S} \cdot 2^{-\lambda}.\end{aligned}$$

Note that it is crucial for PAKE to guarantee *explicit* authentication, otherwise, the token might not be able to detect wrong PIN guesses and then decrement its retries counter to prevent exhaustive PIN guesses.[15] Also note that the PAKE advantage bound may itself include calls to an independent random oracle. PAKE can be instantiated with variants of CPace [23] or SPAKE2 [3, 6] that include explicit authentication. Both protocols were recently considered by the IETF for standardization and CPace was selected in the end.[16] They both meet the required security property, as they have been proved secure in the UC setting which implies the game-based security notion we use [4, 23].

UF $\not\Rightarrow$ SUF-t. Note that one can easily transform sPACA into a protocol that is still UF secure, but not SUF-t secure: similar to CTAP2, let the authenticator generate a global pinToken used as binding states for all its partners and send it (encrypted with the session key output by PAKE) to its partners at the end of Bind executions. This shows that UF does not imply SUF-t.

Performance comparison of CTAP2 and sPACA. It is straightforward to see from Figure 5 and Figure 6 that CTAP2 and sPACA differ mainly in their Bind executions, while sPACA has slightly better performance than CTAP2 in other subprotocols. We therefore compare their performance for binding in terms of message flows, computations (for group exponentiations, hashes, AES) in both sides, and communication complexity, with results summarized in Table 1. There sPACA is instantiated with CPace, which requires 3 flows when explicit authentication is required; while CTAP2 needs 4 flows. Besides, if Bind is executed when the client already has a command to issue, then the last CPace message can be piggy-backed with the authorized command, leading to a very efficient 2-flow binding.[17] As shown in Figure 5, CTAP2 requires two Diffie-Hellman group exponentiations

[15] One does not actually need explicit token-to-client authentication in the proof, as clients do not have long-term secret to protect. This would allow removing the server-side authentication component from the PAKE instantiation for further efficiency. We do not propose to do this and choose to rely on the standard *mutual* explicit authentication property to enable direct instantiation of a standardized protocol.

[16] `https://mailarchive.ietf.org/arch/msg/cfrg/j88r8N819bw88xCOyntuw_Ych-I`

[17] This piggy backing has the extra advantage of associating the end of the binding state with a user gesture by default, which helps detect online dictionary attacks against the token as stated in Section 6.

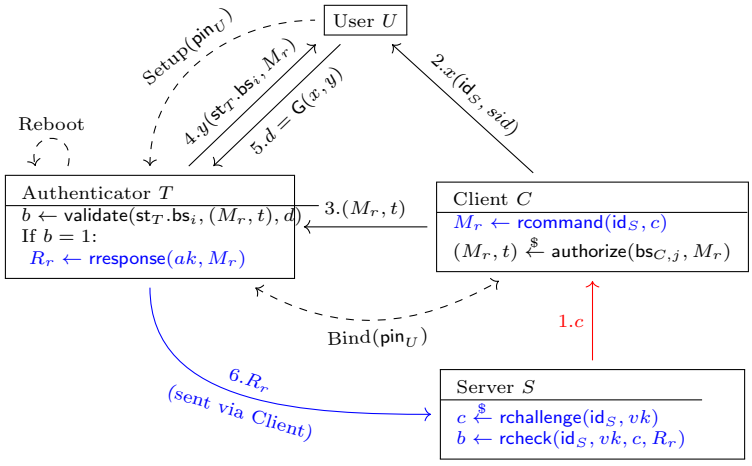

Figure 7: Full PlA+PACA registration flow: black = PACA, blue = PlA, red = authenticated (e.g., TLS), dashed = PACA algorithms/subprotocols.

and $2k$ AES computations (for pt of k-block length) for both parties; the token computes one hash while the client computes two (one for hashing PIN). For sPACA, CPace requires two Diffie-Hellman group exponentiations and four hashes for both sides; the client also needs to compute the PIN hash beforehand. In short, CPace incurs three more hashes but it does not involve AES computations. Note that the most expensive computations are group exponentiations, for which both protocols have two. Regarding communication complexity, both protocols exchange two group elements and two messages with the same length as the binding states, so they are equal if, say, $\kappa = k\lambda = 256$. Overall, sPACA (with CPace) is more efficient than CTAP2. Furthermore, we emphasize that the cryptographic primitives in sPACA could be instantiated with more efficient (but still secure) ones compared to those in CTAP2. For instance, one can use a simple one-time pad (with appropriate key expansion) instead of CBC_0 in Setup to achieve the same SUF security.

9 Composed Security of PlA and PACA

In this section we discuss the composed security of PlA and PACA and the implications of this composition for WebAuthn+CTAP2. The composed protocol, which we refer simply as PlA+PACA is defined in the natural way, and it includes all the parties that appear in Figure 2. We give a typical flow for registration in Figure 7, where we assume PACA authenticator setup and client-to-authenticator binding have been correctly executed. The server role is purely that of a PlA server. The client receives the server challenge via an authenticated channel (i.e., it knows the true server identity id_S when it gets a challenge from the server). It then authorizes the challenge using the PACA protocol and sends it to the authenticator. The authenticator first checks the PACA command (possibly using a user gesture) and, if successful, it produces a PlA response that is conveyed to the server. The flow for authentication looks exactly the same, apart from the fact that the appropriate PlA algorithms are used instead. The requirement on the token is that it supports the combined functionalities of the PACA protocol and the PlA protocol and that it is able to verify the correct authorization of two types of commands, (M_r, t) and (M_a, t), that correspond to PlA registration and authentication. These commands are used to control access to the PlA registration and authentication functionalities. In Appendix E we formally give a syntax for such composed protocols.

The crucial aspect of our security results is that we convey the two-sided authentication guarantees offered by PlA+PACA, and not only the server side guarantees. Indeed, the server-side guarantees

given by the composed protocol are exactly those offered by PlA, as the server is simply a PlA server: if a token was used to register a key, then the server can recognize the same token in authentication. But how does the client and user know which server they are registering at? What guarantee does a user have such that registered credentials cannot be used in a different server? What does a user know about how browser security affects the effectiveness of access control to the token? We answer these questions next.

Security model. We give a very short description of the security model here (the details are in Appendix E). We define a security property called *user authentication (UA)* for the composed protocol. We analyze the PlA+PACA composition in a trust model identical to the PACA model but we require a server-to-client explicit authentication guarantee. This captures a basic guarantee given by TLS, whereby the client knows the true identity of the server that generates the challenge and is ensured the integrity of the received challenge; it allows formalizing explicit server authentication guarantees given to the authenticator and user by the composed protocol. We allow the adversary to create arbitrary bindings between clients and tokens, and to deliver arbitrary commands to these created token sessions. We model server-client interaction via a unified oracle: the adversary can request challenges from server S, via client C aimed at a specific client-token PACA binding. We hardwire the server's true identity, which is justified by our assumption of an authenticated channel between server and client. The token oracles are modeled in the obvious way: if a PACA command is accepted, then it is interpreted as in the PlA security game and the response is given to the adversary. Compromise of binding states and corruption of user pins are modeled as in the PACA security experiment.

Security guarantees. The security goal we define for the composed protocol requires that server session acceptance uniquely identifies honest token sessions, the associated client sessions (or access channels), and the messages exchanged between them, for all the passes in the challenge-response protocol. We show that such security for the composed protocol follows from security of the base protocols, namely server-to-client (TLS), client-to-token (PACA), and token-to-server (PlA) authentication, and from correctness of PlA executions. A corollary is that PlA correctness applies to both registration and authentication and guarantees that server, client and token agree on the server identity.

We now give a short intuition on how we prove this result assuming the underlying PlA and PACA components are secure. Suppose a server authentication session $\pi_S^{i,j}$ accepts, and that the registration session $\pi_S^{i,0}$ used as inputs the attestation public key of token T and the server identity id_S:

- PlA security guarantees the existence of unique partner sessions in T; partnering covers the authentication session and the associated registration session.
- Token sessions are, by construction, created on acceptance of PACA commands. Therefore, a PACA token session must have accepted commands to create the above PlA partner sessions.
- PACA security binds a PACA command accepted by the token to a unique PACA client session (in the SUF/SUF-t corruption model) or to a set of PACA partner client sessions (in the UF/UF-t corruption model).
- PlA security guarantees unique server-side session oracles bound to the token (i.e., they produced a challenge consistent with its view); this implies that the unique client sessions identified above must have produced the PACA commands on input challenges produced by $\pi_S^{i,j}$ and $\pi_S^{i,0}$.

This argument guarantees that unique client and token sessions are bound to the execution of the registration and authentication flows, as we claimed above. If this doesn't hold, then either the PlA protocol can be broken or the PACA protocol can be broken (reduction to the PACA protocol security can be done for the same corruption model).

The details are in Appendix E.

Security in the SUF model. The above result implies that the sPACA protocol from Section 8 composes with WebAuthn to give this security guarantee in the strongest corruption model we considered. Intuitively, no active attacks against the Bind subprotocol can help the attacker beyond the probability of guessing the user PIN (if the attacker does not possess the token to provide a gesture). The corruption of browser windows that have previously been bound to the token may be detected with the help of the user.

Implications for FIDO2. The above result also implies that FIDO2 components WebAuthn+CTAP2 securely compose to give the guarantees above under a weak corruption model UF-t: the protocol is broken if the adversary can corrupt *any* browser window that interacted with the token since the last powerup, or if the adversary can launch an active attack against an honest browser window via the CTAP2 API (i.e., the user thinks it is embedding the PIN but it is actually giving it to the adversary). If the trust model assumed for the client platform excludes such attacks, then WebAuthn+CTAP2 gives the same server-side security guarantees we have detailed above.

User gestures can upgrade security. User authentication gives strong guarantees to the server and client. However, it is not so clear what guarantees it gives to the human user. Clearly there is a guarantee that an attacker that does not control the token cannot force an authentication, as it will be unable to provide a gesture. Furthermore, an attacker that steals the token must still guess the PIN in a small number of tries to succeed in impersonating the user.

One very important aspect of user awareness is dealing with malware attacks that may corrupt browser windows that have been bound to the token. Here, assuming SUF security has been established, the user can be used to prevent the adversary from abusing the binding, provided that the token supports gestures that permit identifying the browser-token session identifier that is issuing each command. In the weaker UF model there is no way to prevent this kind of abuse, as corrupting one binding session allows the adversary to impersonate another binding session.

Gestures can also be used to give explicit guarantees to the user that the server identity used in a PlA session is the intended one. For example, there could be ambiguity with multiple (honest) client windows issuing concurrent commands from multiple servers. Suppose gestures G_r and G_a permit confirming which client session is issuing the registration and authentication commands.[18] In this case we get a strong guarantee that the token registered a credential or authenticated in the server with identifier $\mathsf{id}_S^\star$, where $\mathsf{id}_S^\star$ was explicitly confirmed by the user on the client interface, provided that that binding session (i.e., browser session) issued only one command to the token. Alternatively, G_r and G_a can be defined to directly confirm the specific $\mathsf{id}_S^\star$ value that can be displayed by the authenticator itself and we get the same guarantee.

If the gesture cannot confirm consistency between client and token, then the user will not be able to distinguish which client session (browser window) is issuing the PlA command and know for sure which id_S the command it is approving refers to. However, our composition result does show that trivial gestures are sufficient if the user only establishes one binding session with the token per powerup, as then there is no ambiguity as to which access channel is used and only a single client is providing the server identity as input.

10 Conclusion

We performed the first provable security analysis of the new FIDO2 protocols for a standard for passwordless user authentication. We studied security of FIDO2's core components: WebAuthn and CTAP2, and the overall FIDO2 as their composition. We identified some shortcomings and proposed stronger protocols. We hope that our results will help clarify the security guarantees of the new FIDO2 protocols and help the design and deployment of more secure and efficient passwordless user authentication protocols.

[18] Confirming a client session means that the browser and token somehow display a session identifier that the user can crosscheck and confirm.

References

[1] "FIDO Alliance. Client to authenticator protocol (CTAP) – proposed standard," January 2019, https://fidoalliance.org/specs/fido-v2.0-ps-20190130/fido-client-to-authenticator-protocol-v2.0-ps-20190130.html.

[2] "Google 2-step verification," 2020, https://www.google.com/landing/2step/.

[3] M. Abdalla and M. Barbosa, "Perfect forward security of SPAKE2," Cryptology ePrint Archive, Report 2019/1194, 2019, https://eprint.iacr.org/2019/1194.

[4] M. Abdalla, M. Barbosa, T. Bradley, S. Jarecki, J. Katz, and J. Xu, "Universally composable relaxed password authenticated key exchange," Cryptology ePrint Archive, Report 2020/320, 2020, https://eprint.iacr.org/2020/320.

[5] M. Abdalla, M. Bellare, and P. Rogaway, "The oracle Diffie-Hellman assumptions and an analysis of DHIES," in *Cryptographers Track at the RSA Conference*. Springer, 2001, pp. 143–158.

[6] M. Abdalla and D. Pointcheval, "Simple password-based encrypted key exchange protocols," in *Topics in Cryptology - CT-RSA 2005*, ser. Lecture Notes in Computer Science, A. Menezes, Ed., vol. 3376. Springer, 2005, pp. 191–208. [Online]. Available: https://doi.org/10.1007/978-3-540-30574-3_14

[7] L. b. Amber Gott, "LastPass reveals 8 truths about passwords in the new Password Exposé," https://blog.lastpass.com/2017/11/lastpass-reveals-8-truths-about-passwords-in-the-new-password-expose.html/, November 2017.

[8] M. Bellare, "New proofs for nmac and hmac: Security without collision-resistance," in *CRYPTO 2006*. Springer, 2006, pp. 602–619.

[9] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *CRYPTO 1996*. Springer, 1996, pp. 1–15.

[10] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway, "A concrete security treatment of symmetric encryption," in *Foundations of Computer Science*. IEEE, 1997, pp. 394–403.

[11] M. Bellare, D. Pointcheval, and P. Rogaway, "Authenticated key exchange secure against dictionary attacks," in *International conference on the theory and applications of cryptographic techniques*. Springer, 2000, pp. 139–155.

[12] M. Bellare and P. Rogaway, "Entity authentication and key distribution," in *CRYPTO 1993*. Springer, 1993, pp. 232–249.

[13] ——, "The exact security of digital signatures-how to sign with RSA and Rabin," in *Advances in Cryptology — EUROCRYPT '96*, U. Maurer, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 399–416.

[14] A. Boldyreva, S. Chen, P.-A. Dupont, and D. Pointcheval, "Human computing for handling strong corruptions in authenticated key exchange," in *Computer Security Foundations Symposium (CSF)*. IEEE, 2017, pp. 159–175.

[15] W. W. W. Consortium *et al.*, "Web authentication: An API for accessing public key credentials level 1 – W3C recommendation," March 2019, https://www.w3.org/TR/webauthn.

[16] R. Cramer and V. Shoup, "Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack," *SIAM Journal on Computing*, vol. 33, no. 1, pp. 167–226, 2003.

[17] A. Czeskis, M. Dietz, T. Kohno, D. Wallach, and D. Balfanz, "Strengthening user authentication through opportunistic cryptographic identity assertions," in *ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 404–414. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382240

[18] G. Davis, "The past, present, and future of password security," https://www.mcafee.com/blogs/consumer/consumer-threat-notices/security-world-password-day/, May 2018.

[19] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on information theory*, vol. 29, no. 2, pp. 198–208, 1983.

[20] M. Dworkin, "Recommendation for block cipher modes of operation. methods and techniques," National Inst of Standards and Technology Gaithersburg MD Computer security Div, Tech. Rep., 2001.

[21] FIDO, "Specifications overview," https://fidoalliance.org/specifications/, accessed: 2020-05-21.

[22] I. B. Guirat and H. Halpin, "Formal verification of the W3C web authentication protocol," in *5th Annual Symposium and Bootcamp on Hot Topics in the Science of Security*. ACM, 2018, p. 6.

[23] B. Haase and B. Labrique, "AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2019, no. 2, pp. 1–48, 2019. [Online]. Available: https://doi.org/10.13154/tches.v2019.i2.1-48

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

[25] K. Igoe, D. McGrew, and M. Salter, "Fundamental elliptic-curve Cryptography Algorithms," RFC 6090, Feb. 2011. [Online]. Available: https://rfc-editor.org/rfc/rfc6090.txt

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

[27] T. Jager, S. A. Kakvi, and A. May, "On the security of the PKCS#1 v1.5 signature scheme," in *ACM SIGSAC Conference on Computer and Communications Security, CCS 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. ACM, 2018, pp. 1195–1208. [Online]. Available: https://doi.org/10.1145/3243734.3243798

[28] S. Jarecki, H. Krawczyk, M. Shirvanian, and N. Saxena, "Two-factor authentication with end-to-end password security," in *Public-Key Cryptography - PKC 2018*, 2018, pp. 431–461. [Online]. Available: https://doi.org/10.1007/978-3-319-76581-5_15

[29] K. Moriarty, B. Kaliski, J. Jonsson, and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2," RFC 8017, Nov. 2016. [Online]. Available: https://rfc-editor.org/rfc/rfc8017.txt

[30] B. Nahorney, "Email threats 2017," *Symantec. Internet Security Threat Report*, 2017.

[31] S. H. S. NIST, "Fips pub. 180-2," 2002.

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

[33] O. Pereira, F. Rochet, and C. Wiedling, "Formal analysis of the FIDO 1. x protocol," in *International Symposium on Foundations and Practice of Security*. Springer, 2017, pp. 68–82.

[34] V. Shoup, "Sequences of games: a tool for taming complexity in security proofs." *IACR Cryptol. ePrint Arch.*, vol. 2004, p. 332, 2004.

[35] Verizon, "2017 data breach investigations report," https://enterprise.verizon.com/resources/reports/2017_dbir.pdf, 2017.

A Preliminary Definitions

A.1 Pseudorandom Function

For a function family $F : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^m$, consider the following security experiment associated with an adversary $\mathcal{A}$. In the beginning, sample a bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$, $\mathcal{A}$ is given oracle access, i.e., can make queries, to $F_k(\cdot) = F(k,\cdot)$ where $k \xleftarrow{\$} \{0,1\}^\lambda$. If $b = 1$, $\mathcal{A}$ is given oracle access to $f(\cdot)$ that maps elements from $\{0,1\}^n$ to $\{0,1\}^m$ uniformly at random. In the end, $\mathcal{A}$ outputs a bit b' as a guess of b. The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}^{\mathsf{prf}}_F(\mathcal{A}) = |\Pr[b' = 1|b = 0] - \Pr[b' = 1|b = 1]|$, which measures $\mathcal{A}$'s ability to distinguish F_k (with random k) from a random function f.

F is a *pseudorandom function (PRF)* if: 1) for any $k \in \{0,1\}^\lambda$ and any $x \in \{0,1\}^n$, there exists an efficient algorithm to compute $F(k,x)$; and 2) for any efficient adversary $\mathcal{A}$, $\mathbf{Adv}^{\mathsf{prf}}_F(\mathcal{A})$ is sufficiently small (e.g., roughly $2^{-\lambda}$).

A.2 Collision-Resistant Hash Function Family

Consider a function family $H = \mathcal{K} \times \mathcal{D} \to \mathcal{R}$, where $|\mathcal{D}| > |\mathcal{R}|$ and computing $H_k = H(k,\cdot)$ on any input is efficient given k. In the security experiment, an adversary $\mathcal{A}$ takes as input a random $k \xleftarrow{\$} \mathcal{K}$, then outputs two messages $(x_1, x_2) \in \mathcal{D}$. Its advantage measure $\mathbf{Adv}^{\mathsf{coll}}_H(\mathcal{A})$ is defined as the probability that $x_1 \neq x_2$ and $H_k(x_1) = H_k(x_2)$.

H is *collision-resistant* if for any efficient adversary $\mathcal{A}$, $\mathbf{Adv}^{\mathsf{coll}}_H(\mathcal{A})$ is sufficiently small (e.g., roughly $2^{-\lambda}$). Note that in practice H is not a hash family but consists of only a single or very few hash functions; in this case it should be infeasible to construct an efficient adversary that finds a collision.

A.3 Message Authentication Code

A *message authentication code (MAC)* MAC is a three-tuple $(\mathcal{K}, \mathsf{Mac}, \mathsf{Ver})$:

- $\mathcal{K}$: this is the key space.
- Mac: takes as input a key k and a message m, then outputs a tag t.
- Ver: takes as input a key k, a message m, and a tag t, then outputs a bit b indicating if the tag is valid.

The *correctness* requires that for any $(pk, sk) \xleftarrow{\$} \mathsf{Kg}(1^\lambda)$ and any m, $\mathsf{Ver}(pk, m, \mathsf{Sign}(sk, m)) = 1$.

For security, consider the following security experiment associated with an adversary $\mathcal{A}$. In the beginning, sample a random key $k \xleftarrow{\$} \mathcal{K}$. Then, $\mathcal{A}$ is given access to oracles $\mathsf{Mac}_k(\cdot) = \mathsf{Mac}(k,\cdot)$ and $\mathsf{Ver}_k(\cdot,\cdot) = \mathsf{Ver}(k,\cdot,\cdot)$. In the end, $\mathcal{A}$ outputs a message-tag pair (m,t). Its advantage measure $\mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{MAC}}(\mathcal{A})$ is defined as the probability that $\mathsf{Ver}_k(m,t) = 1$ and m was not queried to the $\mathsf{Mac}_k(\cdot)$ oracle.

MAC is secure in the sense of *existentially unforgeable under chosen message attack (EUF-CMA)* if for any efficient adversary $\mathcal{A}$, $\mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{MAC}}(\mathcal{A})$ is sufficiently small (e.g., roughly $2^{-\log|\mathcal{K}|}$).

A.4 Signature Scheme

A signature scheme Sig consists of three efficient algorithms $(\mathsf{Kg}, \mathsf{Sign}, \mathsf{Ver})$:

- $\mathsf{Kg}(1^\lambda)$: takes as input the security parameter 1^λ and outputs a pair of keys: a public verification key pk and a private signing key sk.
- Sign: takes as input a signing key sk and a message m, then outputs a signature σ.
- Ver: takes as input a verification key pk, a message m, and a signature σ, then outputs a bit b indicating if the signature is valid.

The *correctness* requires that for any $(pk, sk) \xleftarrow{\$} \mathsf{Kg}(1^\lambda)$ and any m, $\mathsf{Ver}(pk, m, \mathsf{Sign}(sk, m)) = 1$.

For security, consider the following security experiment associated with an adversary $\mathcal{A}$. In the beginning, run $(pk, sk) \xleftarrow{\$} \mathsf{Kg}(1^\lambda)$. Then, $\mathcal{A}$ is given pk and access to the oracle $\mathsf{Sign}_{sk}(\cdot) = \mathsf{Sign}(sk, \cdot)$. In the end, $\mathcal{A}$ outputs a message-signature pair (m, σ). Its advantage measure $\mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{Sig}}(\mathcal{A})$ is defined as the probability that $\mathsf{Ver}(pk, m, \sigma) = 1$ and m was not queried to the $\mathsf{Sign}_{sk}(\cdot)$ oracle.

Sig is EUF-CMA-secure if for any efficient adversary $\mathcal{A}$, $\mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{Sig}}(\mathcal{A})$ is sufficiently small (e.g., roughly $2^{-\lambda}$).

A.5 The (Strong) Computational Diffie-Hellman Assumption

Consider a cyclic group $\mathbb{G} = \langle g \rangle$ of prime order q associated with the security parameter λ. The *computational Diffie-Hellman (CDH)* assumption states that it is computationally infeasible to compute g^{ab} given $\mathbb{G}, g, g^a, g^b$ for random $a, b \xleftarrow{\$} \mathbb{Z}_q$. That is, let $\mathbf{Adv}^{\mathsf{cdh}}_{\mathbb{G},g}(\mathcal{A})$ denote the probability that an adversary $\mathcal{A}$ outputs g^{ab}, then we have $\mathbf{Adv}^{\mathsf{cdh}}_{\mathbb{G},g}(\mathcal{A})$ is sufficiently small (e.g., roughly $2^{-\lambda}$) for any efficient adversary $\mathcal{A}$.

For the *strong CDH (sCDH)* assumption [5], an adversary $\mathcal{A}$ is additionally granted oracle access to $\mathcal{O}_a(\cdot, \cdot)$, which takes any group elements $Y, Z \in \mathbb{G}$ as input and checks if $Y^a = Z$. Let $\mathbf{Adv}^{\mathsf{scdh}}_{\mathbb{G},g}(\mathcal{A})$ denote the probability that $\mathcal{A}$ outputs g^{ab}. The sCDH assumption states that for any efficient adversary $\mathcal{A}$, $\mathbf{Adv}^{\mathsf{scdh}}_{\mathbb{G},g}(\mathcal{A})$ is sufficiently small (e.g., roughly $2^{-\lambda}$).

B Password-Authenticated Key Exchange

A *password-authenticated key exchange (PAKE)* protocol is an interactive protocol between two parties (sometimes referred to as client/server or initiator/responder, but in PACA we have client/token). PAKE protocols allow them to establish a high-entropy session key over an insecure channel using only a shared low-entropy, human-memorizable password for mutual authentication.

Since a password has low entropy, an adversary has non-negligible chance of successfully impersonating one of the parties by guessing their shared password. Such an impersonation attack is called an *online* dictionary attack because the adversary cannot mount this attack by itself (referred to as *offline* dictionary attack) but needs to interact with an "online" party to verify its guess. Note that an offline attack, if possible, is disastrous to a PAKE protocol due to the low entropy of the password. Informally, a secure PAKE protocol guarantees that for any efficient adversary an exhaustive online dictionary attack is the best strategy to break the protocol.

We consider the game-based security model for (non-augmented) PAKE protocols with perfect forward secrecy (PFS) and explicit mutual authentication, as described for example in [11], but without explicit separation of passive session executions (i.e., the attacker can adaptively decide whether it is going to be passive or active in a given session). This game-based security model is implied by UC security [4].

Security experiments. Consider an efficient adversary $\mathcal{A}$ associated with the experiments. In the PFS experiment, $\mathcal{A}$ is challenged to distinguish established session keys from truly random ones with an advantage that is better than password guessing. In the explicit authentication experiment, $\mathcal{A}$ is challenged to make a party terminate without having a unique partner (defined later).

The challenger emulates an execution environment in which tokens $T \in \mathcal{T}$ and clients $C \in \mathcal{C}$ communicate using PAKE. The sets of tokens and clients are disjoint, the client initiates the protocol, and each pair of client-token shares a pre-agreed password. The challenger first generates arbitrary global public parameters called the common reference string (CRS) that PAKE may rely on and samples passwords for all pairs of parties from an arbitrary distribution $\mathscr{D}$ (over some password dictionary). Passwords need not be uniformly distributed, but it is assumed that they are sampled independently for each pair of parties. The important parameter of $\mathscr{D}$ is its min-entropy $h_{\mathscr{D}}$, which

intuitively characterizes the most likely password.[19] The challenger manages a set of instances π_P^i, each corresponding to the i-th session instance at party $P \in \mathcal{C} \cup \mathcal{T}$, according to the protocol definition. The adversary $\mathcal{A}$ then takes the CRS as input and interact with the following set of oracles, to which it may place multiple adaptive queries:

SEND: Given a party identity P, an instance i, and a message m, this oracle processes m according to the state of π_P^i (or creates this state if the instance was not yet initialized) and returns any outgoing messages to $\mathcal{A}$.

CORRUPT: Given a pair of party identities (C, T), this oracle returns the corresponding pre-shared password to $\mathcal{A}$.

REVEAL: Given a party identity P and an instance i, this oracle checks π_P^i and, if it has completed as defined by the protocol, the output of the session (usually either a secret key or an abort symbol) is returned.

TEST: Given a party identity P and an instance i, this oracle checks if π_P^i has terminated (i.e., completed without aborting) and it is *fresh* (defined later), if so, challenges the adversary on guessing bit b: if $b = 0$ then the derived key is given to $\mathcal{A}$; otherwise a new independent random key is returned.

Eventually, $\mathcal{A}$ terminates and outputs a guess bit b' in the PFS experiment. As detailed below, trivial attacks are excluded by the notion of session freshness used in the TEST oracle.

Partners and session freshness. Two session instances are *partnered* if their views match with respect to the identity of the peer, exchanged messages, and derived secret keys—the first two are usually interpreted as a session identifier. A session instance is fresh if the instance and its partner(s) (if any) were not queried to TEST or REVEAL before and at least one of the following conditions holds: i) the adversary behaved passively when completing that session; ii) the associated password was not corrupted prior to completion.

Security goals. A PAKE protocol PAKE offers perfect forward secrecy (PFS) if, for any efficient attacker $\mathcal{A}$ interacting with the above experiment, we have

$$|\Pr[b' = 1 | b = 1] - \Pr[b' = 1 | b = 0]| \leq q_s \cdot 2^{-h_{\mathscr{D}}} + \epsilon_{\mathsf{pfs}}\,,$$

where ϵ_{pfs} is a negligible term and q_s is the number of SEND queries in which the adversary delivered a modified message (i.e., actively attacked the session).

A PAKE protocol PAKE provides explicit (mutual) authentication if the following conditions hold, except with probability $q_s \cdot 2^{-h_{\mathscr{D}}} + \epsilon_{\mathsf{ea}}$:

- if a client instance terminates, then it has a unique token partner that has derived the same key and session identifier;
- if a token instance terminates, then it has a unique client partner that has derived the same key and session identifier;

Note that this implies that any active attack launched by $\mathcal{A}$ leads to a session abort with overwhelming probability, unless it was able to guess the password.

We define $\mathbf{Adv}_{\mathsf{PAKE}}(\mathcal{A}, q_s, h_{\mathscr{D}}) = 2q_s \cdot 2^{-h_{\mathscr{D}}} + \epsilon_{\mathsf{pfs}} + \epsilon_{\mathsf{ea}}$.

[19] We use this definition to emphasize that one does not need to assume that the distribution of PINs is uniform; we could just as well assume a small dictionary and uniform sampling, or a more fine-grained definition of advantage considering the sum of the probabilities of the q_s most likely pins.

C IND-1$PA for Deterministic Encryption

In this section, we propose a security notion for deterministic encryption, which we call *indistinguishability under one-time chosen and then random plaintext attack (IND-1$PA)*. This notion is used to analyze CTAP2 security but may be of independent interest when only random messages are encrypted.

Definition 1 (IND-1$PA) *For an arbitrary deterministic encryption scheme* $\mathsf{DE} = (\mathcal{K}, \mathsf{E}, \mathsf{D})$*, consider a security experiment associated with an adversary* $\mathcal{A}$*. The experiment samples* $k \xleftarrow{\$} \mathcal{K}$*,* $b \xleftarrow{\$} \{0,1\}$*, and* $\mathcal{A}$ *does the following:*

1. *Chooses a list of pairs of messages* $\mathcal{L}_a = \langle (m_0^i, m_1^i) \rangle_i$*.*
2. *Samples a list of pairs of random messages* $\mathcal{L}_r = \langle (\tilde{m}_0^i, \tilde{m}_1^i) \rangle_i$*.*
3. *Selects only* one *message pair from* $\mathcal{L}_a$ *and queries its LR encryption oracle:*
   - $\mathcal{O}_{\mathsf{LR}}(m_0, m_1)$*: returns* $\mathsf{E}(k, m_b)$ *if the input messages* m_0, m_1 *are of the same length, or returns* $\perp$ *otherwise.*
4. *Selects any number of message pairs from* $\mathcal{L}_r$ *and queries* $\mathcal{O}_{\mathsf{LR}}(\cdot, \cdot)$*.*

In the end, $\mathcal{A}$ *outputs a bit* b' *as its guess of* b*. Its advantage* $\mathbf{Adv}^{\mathsf{ind\text{-}1\$pa}}_{\mathsf{DE}}(\mathcal{A})$ *is defined as* $|2\Pr(b = b') - 1|$ *or equivalently* $|\Pr[b' = 1 | b = 0] - \Pr[b' = 1 | b = 1]|$*.*

Relation to one-time IND-CPA. IND-1$PA clearly implies *one-time* IND-CPA for which only one $\mathcal{O}_{\mathsf{LR}}$ query is allowed. One-time IND-CPA was defined as security against passive attacks by Cramer and Shoup [16]. We use $\mathbf{Adv}^{\mathsf{ot\text{-}ind\text{-}cpa}}_{\mathsf{DE}}(\mathcal{A})$ to denote its advantage measure. Note that one-time IND-CPA does not imply IND-1$PA, e.g., the one-time pad is one-time-IND-CPA-secure but not IND-1$PA-secure.

D Security Proofs

Our proofs follow the game-playing technique [34] that considers a sequence of games (i.e., security experiments) associated with an adversary. Each pair of consecutive games are "close" enough such that the *difference* of the adversary's "winning" probabilities in the two games can be properly bounded.

D.1 Proof of Theorem 1

Proof: Consider a sequence of games (i.e., security experiments) and let $\Pr_i, i \geq 0$ denote the winning probability of $\mathcal{A}$ in **Game** i.

Game 0: This is the original PlA experiment, so $\Pr_0 = \mathbf{Adv}^{\mathsf{pla}}_{\mathrm{WebAuthn}}(\mathcal{A})$.

Game 1: This game is identical as Game 0, except that it aborts if a H hash collision occurs. It is straightforward to construct an efficient adversary $\mathcal{B}$ against the collision-resistance security of H such that $|\Pr_0 - \Pr_1| \leq \mathbf{Adv}^{\mathsf{coll}}_{\mathsf{H}}(\mathcal{B})$.

Game 2: This game aborts if there exist two identical challenges. We have $|\Pr_1 - \Pr_2| \leq q_{\mathsf{S}}^2 \cdot 2^{\lambda}$ due to the random c. Note that at this point we excluded any possibility of session identifier colliding on the server side.

Game 3: This game aborts if a server-side oracle accepts without having a unique token-side partner. This bad event can be reduced to an efficient adversary $\mathcal{C}$ against the EUF-CMA security of the signature scheme Sig as follows.

$\mathcal{C}$ first guesses the offending session $\pi_S^{i,j}$ with probability at least $1/q_{\mathsf{S}}$, then simulates the game by answering all queries related to that credential with the signing oracle $\mathsf{Sig.Sign}_{sk}(\cdot)$ and the associated public key. There are two cases, $\pi_S^{i,j}$ has at least two partners or no partner.

Let us consider first the case where $j = 0$. Having two partners means that two token-side registration sessions signed the same message; this can happen with probability at most ${q_{\mathsf{C}}}^2 \cdot 2^{-\lambda}$ due to the random *cid*. On the other hand, if there is no partner, then $\mathcal{C}$ has forged a valid signature and wins the EUF-CMA game. Let us now consider the case where $j > 0$. Here we know the server oracle $\pi_S^{i,j}$ accepts implies that $\pi_S^{i,0}$ has successfully verified a signature on a public-key pk that was established in the registration session. From the previous case, we know $\pi_S^{i,0}$ is uniquely partnered with a token oracle $\pi_T^{k,0}$. By the definition of partnership, if $\pi_S^{i,j}$ has two partners, they must all belong to the authentication sessions following $\pi_T^{k,0}$; but this is impossible because the signature counter n is incremented for every new authentication session. On the other hand, if $\pi_S^{i,j}$ has no partner (note again that we already established that $\pi_S^{i,0}$ must have a unique partner) but still accepts, then $\mathcal{C}$ has forged a valid signature and wins the EUF-CMA game.

Therefore, we have $|\Pr_2 - \Pr_3| \leq q_{\mathsf{C}} \cdot \mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{Sig}}(\mathcal{C}) + {q_{\mathsf{C}}}^2 \cdot 2^{-\lambda}$.

Final analysis. At this point the attacker has probability 0 of succeeding, since unique partnering is guaranteed and we have ruled out collisions on the session identifier on the server side. □

D.2 IND-1$PA Security of CBC_0

In the following theorem, we prove that CBC_0 achieves IND-1$PA security defined in Appendix C by modeling its underlying AES-256 cipher E as a PRF. This implies that CBC_0 also achieves one-time IND-CPA security, which allows only one chosen message pair $|\mathcal{L}_a| = 1$ and no random message pair $|\mathcal{L}_r| = 0$.

Theorem 4 *For any efficient adversary $\mathcal{A}$, there exists an efficient adversary $\mathcal{B}$ such that:*

$$\mathbf{Adv}^{\mathsf{ind\text{-}1\$pa}}_{\mathsf{CBC}_0}(\mathcal{A}) \leq 2\mathbf{Adv}^{\mathsf{prf}}_{E}(\mathcal{B}) + 2\left(|\mathcal{L}_a||\mathcal{L}_r| + \binom{|\mathcal{L}_r|}{2} + \binom{\frac{\mu}{\lambda}}{2}\right)2^{-\lambda},$$

where $\lambda = 128$ denotes the block size and μ denotes the total bit length of ciphertexts in response to all encryption oracle queries.

Proof: Consider a sequence of games (i.e., security experiments) and let $\Pr_i$ denote the probability that $\mathcal{A}$ outputs $b' = 1$ in **Game** i for $i \geq 0$.

Game 0: This is the left world of the original IND-1$PA experiment (i.e., $b = 0$), so $\Pr_0 = \Pr[b' = 1|b = 0]$.

Game 1: This game replaces the PRF E with a random function $f : \{0,1\}^\lambda \to \{0,1\}^\lambda$. Note that it is straightforward to simulate either Game 0 or Game 1 with the PRF oracles either E_k (for random k) or f. Therefore, there exists an efficient adversary $\mathcal{B}_0$ against the PRF security of E such that $|\Pr_0 - \Pr_1| \leq \mathbf{Adv}^{\mathsf{prf}}_{E}(\mathcal{B}_0)$.

Game 2: This game aborts if any of the left messages in $\mathcal{L}_a$ shares the same first block with any of the left messages in $\mathcal{L}_r$ or if any two left messages in $\mathcal{L}_r$ share the same first block. Since messages in $\mathcal{L}_r$ are chosen independently at random, each collision happens with probability $2^{-\lambda}$. By a union bound, we have $|\Pr_1 - \Pr_2| \leq (|\mathcal{L}_a||\mathcal{L}_r| + \binom{|\mathcal{L}_r|}{2}) \cdot 2^{-\lambda}$.

Game 3: This is the "mirror" game of Game 2, where the LR encryption oracle always encrypts the right message (i.e., $b = 1$). Note that Game 2 and Game 3 behave identically as long as all f inputs are *distinct* in both games. Therefore, it is sufficient to bound the probability of input collisions. Since there are μ/λ blocks in each world, by a union bound we have $|\Pr_2 - \Pr_3| \leq 2\binom{\mu/\lambda}{2}2^{-\lambda}$.

Game 4: This is the "mirror" game of Game 1. With the same argument, we have $|\Pr_3 - \Pr_4| \leq (|\mathcal{L}_a||\mathcal{L}_r| + \binom{|\mathcal{L}_r|}{2}) \cdot 2^{-\lambda}$.

Game 5: This is the real right world of the IND-1$PA experiment, so $\Pr_5 = \Pr[b' = 1|b = 1]$. Similar to the reasoning in Game 1, there exists an efficient adversary $\mathcal{B}_1$ such that $|\Pr_4 - \Pr_5| \leq \mathbf{Adv}^{\mathsf{prf}}_E(\mathcal{B}_1)$.

Final analysis: The proof is concluded by noticing that $\mathbf{Adv}^{\mathsf{ind\text{-}1\$pa}}_{\mathsf{CBC}_0}(\mathcal{A}) = |\Pr[b' = 1|b = 0] - \Pr[b' = 1|b = 1]| = |\Pr_0 - \Pr_5|$.

We also refer to [10] (Theorem 16) for a similar security analysis of the randomized CBC scheme that can be adapted to our case. □

D.3 Proof of Theorem 2

Proof: Consider a sequence of games (i.e., security experiments) and let $\Pr_i$ denote the winning probability of $\mathcal{A}$ in **Game** i for $i \geq 0$.

Game 0: This is the original UF-t experiment, so $\Pr_0 = \mathbf{Adv}^{\mathsf{uf\text{-}t}}_{\mathrm{CTAP2}}(\mathcal{A})$.

Game 1: This game replaces all keys $K = \mathcal{H}_1(abG.x) = \mathcal{H}_1(baG.x)$ established in Setup and $\mathsf{Execute}$ queries with independent random values $\widetilde{K} \xleftarrow{\$} \{0,1\}^{2\lambda}$. We apply a hybrid argument to replace these $q_{\mathsf{S}} + q_{\mathsf{E}}$ keys one by one by reducing to the sCDH security of $(\mathbb{G}, G)$, then by a union bound there exists an efficient adversary $\mathcal{B}$ such that $|\Pr_0 - \Pr_1| \leq (q_{\mathsf{S}} + q_{\mathsf{E}}) \cdot \mathbf{Adv}^{\mathsf{scdh}}_{\mathbb{G},G}(\mathcal{B})$.

To simulate the hybrid game that replaces the k-th key with a random value, we consider an efficient sCDH adversary $\mathcal{B}_k$ that embeds the challenge group elements $\tilde{a}G, \tilde{b}G$ respectively into (all session oracles of) the corresponding token T and the corresponding client oracle π^j_C. $\mathcal{B}_k$ answers queries to the random oracle $\mathcal{H}_1$ via *lazy sampling*, where it samples independent random output values for distinct queries while ensuring consistency (i.e., the same query is answered with the same value). Note that when some oracle π^i_T receives a malicious bG sent by $\mathcal{A}$ via a Send query, $\mathcal{B}_k$ may not know the $\tilde{a}bG.x$ value but still has to check if $\mathcal{H}_1(\tilde{a}bG.x)$ has been queried before to ensure sampling consistency. In this case, $\mathcal{B}_k$ can check with its sCDH verification oracle $\mathcal{O}_{\tilde{a}}(\cdot,\cdot)$ (where $\mathcal{O}_{\tilde{a}}(bG, Z)$ checks if $\tilde{a}bG = Z$); it can also answer $\mathcal{H}_1(\tilde{a}\tilde{b}G.x)$ in this way if needed. The rest of the hybrid game for $\mathcal{A}$ can be easily simulated. By the design of hybrid games, two consecutive games proceed identically unless $\mathcal{A}$ ever queries $\mathcal{H}_1(\tilde{a}\tilde{b}G.x)$; if so $\mathcal{B}_k$ can win the sCDH game (by checking if $\mathcal{O}_{\tilde{a}}(\tilde{b}G, \cdot) = 1$ for all $\mathcal{A}$'s $\mathcal{H}_1$ queries).

Game 2: This game aborts if a hash collision occurs for the truncated hash function H'. It is straightforward to construct an efficient adversary $\mathcal{C}$ such that $|\Pr_1 - \Pr_2| \leq \mathbf{Adv}^{\mathsf{coll}}_{\mathsf{H}'}(\mathcal{C})$.

Game 3: This game replaces all authentication tags $t_p = \mathcal{H}_2(\widetilde{K}, c_p)$ computed in Setup queries with independent random values $\tilde{t}_p \leftarrow \{0,1\}^{\lambda}$. By applying a hybrid argument, we can replace these tags one by one. Then, two consecutive hybrid games proceed identically unless $\mathcal{A}$ ever queries $\mathcal{H}_2(\widetilde{K}, c_p)$, which happens with probability at most $2^{-2\lambda}$ for each $\mathcal{H}_2$ query since $\widetilde{K}$ is independently random. Therefore, by a union bound we have $|\Pr_2 - \Pr_3| \leq q_{\mathsf{S}} q_{\mathcal{H}} \cdot 2^{-2\lambda}$.

Game 4: This game replaces all encrypted PINs $c_p = \mathsf{E}(\widetilde{K}, \mathsf{pin}_U)$ computed in Setup queries with the encryptions $\tilde{c}_p = \mathsf{E}(\widetilde{K}, \mathsf{pin}_0)$ of an arbitrarily fixed PIN $\mathsf{pin}_0 \in \mathcal{PIN}$. Again, we need to apply a hybrid argument to replace them one by one. It is straightforward to construct an efficient adversary against the one-time IND-CPA security of CBC_0 that perfectly simulates two consecutive hybrid games using its LR encryption oracle. By a union bound, there exists an efficient adversary $\mathcal{E}$ such that $|\Pr_3 - \Pr_4| \leq q_{\mathsf{S}} \mathbf{Adv}^{\mathsf{ot\text{-}ind\text{-}cpa}}_{\mathsf{CBC}_0}(\mathcal{E})$.

Since an encrypted PIN is of length 4 blocks, according to Theorem 4 we have $\mathbf{Adv}^{\mathsf{ot\text{-}ind\text{-}cpa}}_{\mathsf{CBC}_0}(\mathcal{E}) \leq 2\mathbf{Adv}^{\mathsf{prf}}_E(\mathcal{D}_1) + 12 \cdot 2^{-\lambda}$ for some efficient adversary $\mathcal{D}_1$.

Game 5: This game replaces all encrypted PIN hashes $c_{ph} = \mathsf{E}(\widetilde{K}, \mathsf{H}'(\mathsf{pin}_U))$ and encrypted pinTokens $c_{pt} = \mathsf{E}(\widetilde{K}, pt)$ computed in $\mathsf{Execute}$ queries with respectively $\tilde{c}_{ph} = \mathsf{E}(\widetilde{K}, \mathsf{H}'(\mathsf{pin}_0))$ and $\tilde{c}_{pt} = \mathsf{E}(\widetilde{K}, \widetilde{pt})$ (for an independent random $\widetilde{pt} \xleftarrow{\$} \{0,1\}^{k\lambda}$). As illustrated below, we can construct an efficient adversary against the IND-1$PA security of CBC_0 that perfectly simulates the consecutive hybrid

games. Then, by a union bound, there exists an efficient adversary $\mathcal{F}$ such that $|\Pr_4 - \Pr_5| \leq q_{\mathsf{E}}\mathbf{Adv}^{\mathsf{ind\text{-}1\$pa}}_{\mathsf{CBC}_0}(\mathcal{F})$.

To simulate the consecutive hybrid games that replaces c_{ph}, c_{pt} with $\tilde{c}_{ph}, \tilde{c}_{pt}$ in the k-th $\mathsf{Execute}$ query, we consider an efficient adversary $\mathcal{F}_k$ against the IND-1\$PA security of CBC_0. After all user PINs are sampled, $\mathcal{F}_k$ sets $\mathcal{L}_a = \langle(\mathsf{pin}_U, \mathsf{pin}_0)\rangle_{U\in\mathcal{U}}$ and sets $\mathcal{L}_r = \langle(pt_i, \widetilde{pt}_i)\rangle_i$ as q_{R} pairs of random $k\lambda$-bit strings. Then, $\mathcal{F}_k$ uses $\langle pt_i\rangle_i$ as the pinTokens. In the k-th $\mathsf{Execute}(\pi^i_T, \pi^j_C)$ query, say pin_U is used to set up T and pt_i is used as π^i_T's pinToken; $\mathcal{F}_k$ queries its LR encryption oracle with the message pairs $(\mathsf{pin}_U, \mathsf{pin}_0)$ and $(pt_i, \widetilde{pt}_i)$ and then uses the answers as the encrypted PIN hash and the encrypted pinToken respectively. In this way, $\mathcal{F}_k$ can simulate the hybrid games perfectly, where in the left world it simulates the hybrid game with c_{ph}, c_{pt} and in the right world it simulates the hybrid game with $\tilde{c}_{ph}, \tilde{c}_{pt}$.

Now, note that $|\mathcal{L}_a| = |\mathcal{U}|, |\mathcal{L}_r| = q_{\mathsf{R}}$, an encrypted PIN hash is of 1-block length and an encrypted pinToken is of k-block length. According to Theorem 4 we have $\mathbf{Adv}^{\mathsf{ind\text{-}1\$pa}}_{\mathsf{CBC}_0}(\mathcal{F}) \leq 2\mathbf{Adv}^{\mathsf{prf}}_E(\mathcal{D}_2) + 2(|\mathcal{U}|q_{\mathsf{R}} + \binom{q_{\mathsf{R}}}{2} + \binom{k+1}{2}) \cdot 2^{-\lambda} \leq 2\mathbf{Adv}^{\mathsf{prf}}_E(\mathcal{D}_2) + (2|\mathcal{U}|q_{\mathsf{R}} + q^2_{\mathsf{R}} + (k+1)^2) \cdot 2^{-\lambda}$ for some efficient adversary $\mathcal{D}_2$.

Game 6: This game aborts if there exists a token oracle π^i_T that accepts a malicious c_{ph} sent by $\mathcal{A}$ via a Send query, i.e., the decrypted PIN hash matches $\mathsf{st}_T.\mathsf{s}$. Note that after Game 5 a token's stored PIN hash is independent from $\mathcal{A}$'s view if $\mathcal{A}$ does not corrupt the corresponding user PIN, because all exchanged messages are independent of the user PIN and the UF-t adversary $\mathcal{A}$ is not allowed to launch active attacks against clients (that input the user PIN) with $\mathsf{Connect}$ queries. Since each Setup query allows $\mathcal{A}$ to set up a token on which it is able to make at most 8 guesses (or 3 guesses without involving user interaction) about the token's secret PIN hash and each guess succeeds with probability at most $2^{-h_{\mathscr{D}}}$ (note that after Game 2 distinct PINs generate distinct PIN hashes), by a union bound we have $|\Pr_5 - \Pr_6| \leq 8q_{\mathsf{S}} \cdot 2^{-h_{\mathscr{D}}}$.

Final analysis: In Game 6, any token T's pinToken pt is independent of $\mathcal{A}$'s view before it is used to answer $\mathsf{Authorize}$ or $\mathsf{Validate}$ queries, if $\mathcal{A}$ does not corrupt the user PIN or compromise any of T's partners. By modeling HMAC' as a random oracle $\mathcal{H}_2$, $\mathsf{Authorize}$ and $\mathsf{Validate}$ queries do not reveal information about pt either. In order to forge a valid tag $t = \mathcal{H}_2(pt, M)$, $\mathcal{A}$ can either query $\mathcal{H}_2$ by guessing pt correctly or guess t directly, which together happens with probability at most $2^{-k\lambda} + 2^{-\lambda}$ for each $\mathsf{Validate}$ query. By a union bound we have $\Pr_6 \leq q_{\mathsf{V}} \cdot (2^{-k\lambda} + 2^{-\lambda})$.

□

D.4 UF-t Security of CTAP2*

Theorem 5 *Let $\mathscr{D}$ be an arbitrary distribution over $\mathcal{PIN}$ with min-entropy $h_{\mathscr{D}}$. For any efficient adversary $\mathcal{A}$ making at most $q_{\mathsf{S}}, q_{\mathsf{E}}, q_{\mathsf{R}}$ queries respectively to* $\mathsf{Setup}, \mathsf{Execute}, \mathsf{Reboot}$*, there exist efficient adversaries $\mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}$ such that:*

$$\begin{aligned}\mathbf{Adv}^{\mathsf{uf\text{-}t}}_{\mathrm{CTAP2}}(\mathcal{A}) \leq\ & 8q_{\mathsf{S}} \cdot 2^{-h_{\mathscr{D}}} + (q_{\mathsf{S}} + q_{\mathsf{E}})\mathbf{Adv}^{\mathsf{scdh}}_{\mathbb{G},G}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{coll}}_{\mathsf{H}'}(\mathcal{C}) \\ & + 2(q_{\mathsf{S}} + q_{\mathsf{E}})\mathbf{Adv}^{\mathsf{prf}}_{\mathrm{AES}-256}(\mathcal{D}) + q_{\mathsf{R}}\mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{HMAC}'}(\mathcal{E}) \\ & + (12q_{\mathsf{S}} + 2|\mathcal{U}|q_{\mathsf{R}}q_{\mathsf{E}} + q^2_{\mathsf{R}}q_{\mathsf{E}} + (k+1)^2q_{\mathsf{E}}) \cdot 2^{-\lambda}.\end{aligned}$$

Proof: This proof is almost the same as the proof of Theorem 2 in Appendix D.3, except that there is no Game 3 and the final analysis is concluded by reducing to the EUF-CMA security of HMAC' as follows.

Final analysis: In this final game, any token T's pinToken pt is independent of $\mathcal{A}$'s view before it is used to answer $\mathsf{Authorize}$ and $\mathsf{Validate}$ queries, if $\mathcal{A}$ does not corrupt the associated user PIN or compromise any of T's partners. Consider an efficient adversary $\mathcal{E}$ against the EUF-CMA security of HMAC'. $\mathcal{E}$ first guesses the target reboot cycle (that corresponds to the target token oracle π^i_T that accepts a tag forged by $\mathcal{A}$) with probability at least $1/q_{\mathsf{R}}$, then uses its Mac_k and Ver_k oracles to

answer $\mathcal{A}$'s Authorize and Validate queries to T's partners and T's session oracles that share the same target pinToken. In this way, $\mathcal{E}$ can perfectly simulate $\mathcal{A}$'s view without knowing the target pinToken because in the final game pinTokens are only used in answering Authorize and Validate queries. If $\mathcal{E}$ guesses the reboot cycle correctly, then $\mathcal{A}$ wins implies that $\mathcal{E}$ wins. Therefore, we can bound $\mathcal{A}$'s winning probability by $q_{\mathsf{R}}\mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{HMAC}'}(\mathcal{E})$. □

D.5 Proof of Theorem 3

Proof: Consider a sequence of games and let $\Pr_i, i \geq 0$ denote the winning probability of $\mathcal{A}$ in **Game** i.

Game 0: This is the original SUF experiment, so $\Pr_0 = \mathbf{Adv}^{\mathsf{suf}}_{\mathrm{sPACA}}(\mathcal{A})$.

Game 1: This game replaces all shared keys $K = \mathcal{H}(abG.x)$ established in Setup queries with independent random values $\widetilde{K} \xleftarrow{\$} \{0,1\}^\lambda$. Since ECDH is executed only in Setup executions, we no longer requires the sCDH assumption. Similar to Game 1 in Appendix D.3, there exists an efficient adversary $\mathcal{B}$ against the CDH security of $(\mathbb{G}, G)$ such that $|\Pr_0 - \Pr_1| \leq q_{\mathsf{S}}\mathbf{Adv}^{\mathsf{cdh}}_{\mathbb{G},G}(\mathcal{B})$.

Game 2 and **Game 3** are the same as Game 2 and Game 4 in Appendix D.3, so there exist efficient adversary $\mathcal{C}$ and $\mathcal{D}$ such that the game differences are bounded by $\mathbf{Adv}^{\mathsf{coll}}_{\mathsf{H}'}(\mathcal{C}) + 2q_{\mathsf{S}}\mathbf{Adv}^{\mathsf{prf}}_{E}(\mathcal{D}) + 12q_{\mathsf{S}} \cdot 2^{-\lambda}$.

A PAKE attacker. We now describe an efficient PAKE adversary $\mathcal{E}$ that we will use to bridge the following two game hops. The adversary is parameterized with a switch bit $c \in \{\top, \bot\}$ that instructs it to attack either explicit authentication (which we use to hop to Game 4) or perfect forward secrecy (which we use to hop to Game 5) of PAKE, as defined in Appendix B.

First, we note that to simulate PACA games, the PAKE challenger samples passwords by first sampling a PIN from $\mathcal{PIN}$ according to distribution $\mathscr{D}$ then hashing it with H'. After Game 2 there are no H' collisions, so the resulting password distribution has the same min-entropy as $\mathscr{D}$. Since $\mathcal{PIN}$ has low entropy, $\mathcal{E}$ can brute-force the PIN hashes offline and record all $(\mathsf{pin}, \mathsf{H}'(\mathsf{pin}))$ pairs for PACA simulation. Note that PAKE clients essentially correspond to PACA users since PACA clients do not own PINs but only input them; PACA clients execute PAKE on behalf of the users. $\mathcal{E}$ can simulate Execute, Connect, Send using its SEND oracle; after Game 3, Reboot, Setup can be simulated without oracle queries. To simulate a Corrupt query, $\mathcal{E}$ first gets the PIN hash using its CORRUPT oracle, then looks up in its recorded PIN-hash pairs for the corresponding PIN. Reveal queries are simply forwarded to REVEAL. To simulate Authorize, Validate queries, $\mathcal{E}$ needs to get the secret binding states (i.e., PAKE session keys). It performs differently depending on the switch bit c: if $c = \top$, $\mathcal{E}$ queries its REVEAL oracle; if $c = \bot$, $\mathcal{E}$ queries its TEST oracle. When $\mathcal{A}$ terminates: if $c = \top$, $\mathcal{E}$ wins if explicit authentication is ever broken in the PAKE game; if $c = \bot$, $\mathcal{E}$ returns 1 if $\mathcal{A}$ won the PACA game and 0 otherwise.

We now analyze the maximum number of malicious SEND queries placed by this adversary. To compute this bound we assume that PAKE is three-pass, with the client as the initiator, i.e., Connect triggers the first pass, each client oracle accepts at most one Send query, and each token oracle accepts at most two Send queries. This applies to all of our suggested PAKE instantiations. Since each token allows $\mathcal{A}$ to make at most 8 failed binding attempts (or 3 guesses without requiring user interaction) about the password (i.e., $\mathsf{H}'(\mathsf{pin}_U)$), we know that $\mathcal{E}$ will make at most $16q_{\mathsf{S}}$ queries to its SEND oracle in order to deal with token-side Send queries. Furthermore, $\mathcal{A}$ has to make a Connect query before interacting with a client, which means that $\mathcal{E}$ makes at most $2q_{\mathsf{C}}$ SEND queries to deal with client-side Send queries by $\mathcal{A}$. In total $\mathcal{E}$ makes at most $16q_{\mathsf{S}} + 2q_{\mathsf{C}}$ SEND queries where it actively attacks the session.

Game 4: This game aborts if the explicit authentication guarantee is violated on either the client side or the token side. We bound the game difference in this hop by running the above adversary $\mathcal{E}$ in its authentication adversary ($c = \top$) mode. Then, we have $|\Pr_3 - \Pr_4| \leq (16q_{\mathsf{S}} + 2q_{\mathsf{C}}) \cdot 2^{-h_{\mathscr{D}}} + \epsilon_{\mathsf{ea}}$.

Game 5: This game replaces all PAKE session keys for which the password was not corrupted prior to their completion with independent random values. We bound the game difference in this hop by

running the above adversary $\mathcal{E}$ in its key secrecy adversary ($c = \perp$) mode. Note that $\mathcal{E}$ perfectly simulates the two games and hence $|\Pr_4 - \Pr_5| \leq (16q_{\mathsf{S}} + 2q_{\mathsf{C}}) \cdot 2^{-h_{\mathscr{D}}} + \epsilon_{\mathsf{pfs}}$.

Final analysis: This is similar to the final analysis in Appendix D.4. In this final game, binding states (i.e., PAKE session keys) of the accepting token oracle π_T^i and its unique partner π_C^j (guaranteed by PAKE client authentication) are independent of $\mathcal{A}$'s view before they are used to answer Authorize and Validate queries, if $\mathcal{A}$ does not corrupt the associated user PIN or compromise π_C^j. Consider an efficient adversary against the EUF-CMA security HMAC'. $\mathcal{F}$ first guesses the accepting token oracle π_T^i with probability at least $1/(q_{\mathsf{E}} + q_{\mathsf{C}})$, then simulates the game with its Mac_k and Ver_k oracles. $\mathcal{A}$ wins implies $\mathcal{F}$ wins. Therefore, we have $|\Pr_4 - \Pr_5| \leq (q_{\mathsf{E}} + q_{\mathsf{C}}) \cdot \mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{HMAC}'}(\mathcal{F})$.

□

E Formal Composition Result for PlA+PACA

In this section we discuss the combined security of PlA and PACA as a black box, and then analyze the implications for the WebAuthn+CTAP2 and WebAuthn+sPACA instantiations. The composed protocol, which we refer simply as PlA+PACA is defined in the natural way, and it includes all the parties that appear in Figure 2. The syntax for the protocol is identical to that of PlA, with the single difference that commands (i.e., transformed challenges) sent by clients to the token must be authenticated using the PACA protocol. This leads to the following syntax.

E.1 Protocol Syntax

The state of authenticator T, denoted st_T is partitioned into the following components: i) static PlA attestation key pair ($\mathsf{st}_T.ak, \mathsf{st}_T.vk$); ii) static PlA registration contexts $\mathsf{st}_T.\mathsf{rct}$; iii) static PACA storage $\mathsf{st}_T.\mathsf{ss}$; iv) volatile PACA state $\mathsf{st}_T.\mathsf{vs}$; A client C may have multiple binding states, which we denote by $\mathsf{bs}_{C,j}$. A server S has its static registration contexts rcs.

A PlA+PACA protocol consists of a the following algorithms and subprotocols, all of which can be executed a number of times, except if stated otherwise:

Key Generation This algorithm is executed *at most once* for each authenticator; it generates an attestation key pair (ak, vk) using key generation algorithm Kg.

Reboot This algorithm represents a powerdown/powerup cycle and it is executed by the authenticator. We will use $\mathsf{st} \xleftarrow{\$} \mathsf{reboot}(\mathsf{st.ss}, (\mathsf{st}.ak, \mathsf{st}.vk), \mathsf{st.rct})$ to denote the execution of this algorithm; intuitively it will erase all volatile storage.

Setup This subprotocol is executed *at most once* for each authenticator. No prior state is assumed for any of the participants. The user inputs a PIN through the client. At the end of execution, the authenticator initializes its static storage state $\mathsf{st.ss}$ according to the protocol and resets all other parts of the state. Static storage is read-only for all other algorithms and subprotocols. The client (and through it the user) gets an indication of whether the protocol completed successfully.

Bind This subprotocol is executed by the user, client and authenticator parties to establish a secure session over which commands can be issued. The user inputs its PIN through the client, whereas the token inputs its static storage and powerup states. At the end of this phase, in the case of success both the authenticator and the client get a new binding state; the authenticator may update its powerup state (e.g., a counter).[20] If the protocol fails, no binding states are set, but the authenticator powerup state may still be updated. We assume the client always initiates this protocol once it gets the PIN from the user.

[20] When such an update is possible, and to avoid concurrency issues, it is natural to assume that the token excludes concurrent executions of the binding protocol.

Register This subprotocol is executed between the four parties. The server inputs a server identity id_S and a set of attestation public keys; the client inputs a server identity $\hat{\mathsf{id}}_S$ and a binding state $\mathsf{bs}_{C,j}$; and the authenticator inputs a binding state $\mathsf{st}_T.\mathsf{bs}_i$ and its attestation key pair. The user inputs a gesture G_r. At the end of the protocol, if this is successful, the token and the server obtain new registration contexts, which may be different. Note that the token may successfully complete the protocol, and the server may fail to, in the same run.

Authenticate This subprotocol is executed between the four parties. The server inputs its registration contexts rcs; the client inputs a server identity $\bar{\mathsf{id}}_S$ and a binding state $\mathsf{bs}_{C,j}$; and the authenticator inputs a binding state $\mathsf{st}_T.\mathsf{bs}_i$ and its registration contexts. The user inputs a gesture G_a. At the end of the protocol, the server accepts or rejects.

Correctness is defined as PlA correctness, extended with a notion of correct setup of the PACA binding states. Intuitively, PlA correctness should hold for any sequence of executions that guarantees that the commands sent to the token for registration and authentication are issued under conditions covered by PACA correctness. We omit a formal definition.

A restricted class of protocols. We restrict our attention to protocols that combine a PlA and a PACA protocol in a specific black-box way, which we describe next. This captures our target applications WebAuthn+CTAP2 and it allows us to have a more intuitive security definition. It is clear from the syntax above that PlA+PACA inherits without change the Key Generation algorithm from PlA and the Setup, Reboot, and Bind subprotocols from PACA. The interaction between the PlA and PACA algorithms is visible only in the Register and Authenticate subprotocols, which like for PlA we restrict to the following two-pass protocol that uses PlA and PACA:

- Server-side computation is split into four procedures: $\mathsf{rchallenge}$ and rcheck for registration, $\mathsf{achallenge}$ and acheck for authentication. The challenge algorithms are probabilistic, take the server's input to either the Register or Authenticate subprotocol and output a challenge. The check algorithms get the same inputs, the challenge, and a response, and output accept or reject. The registration check additionally outputs a registration context rcs, encoding server and user identities. This is identical to PlA.
- Client-side computation extends PlA by authenticating the outgoing commands with PACA authentication. More precisely, the client first convert the challenge into a PlA command using $\mathsf{rcommand}$ or $\mathsf{acommand}$, encoding the results as M_r and M_a, respectively. Finally it uses PACA to obtain an authorized command that can be sent to the token.
- Authenticator-side computation does the obvious thing: on input an authorized command of the form (M_r, t) or (M_a, t) it first verifies the authenticity of the command. If successful, then uses the appropriate PlA algorithm $\mathsf{rresponse}$ or $\mathsf{aresponse}$ that, on input a command M to generate a response. In the case of registration, this algorithm also outputs a registration context rct, encoding a server identity and optionally a user identity.

E.2 Security Model

We now discuss the security model in which we will analyze this protocol. The trust model is identical to the PACA model but we add a server-to-client explicit authentication guarantee that does not exist in the PlA setting. This captures a basic guarantee given by TLS, whereby the client knows the true identity of the server that generates the challenge and is ensured the integrity of the received challenge; it allows capturing the explicit server authentication guarantees given to the authenticator and user by the composed protocol.

Session oracles. We keep track of three types of token sessions: volatile PACA binding sessions π_T^i, static PlA registration sessions $\pi_T^{i,0}$ and volatile PlA authentication sessions $\pi_T^{i,j}$ for $j > 0$. We also keep track of client PACA binding oracles π_C^j and their corresponding binding states $\pi_C^j.\mathsf{bs}$. Finally, server oracles are structured as in PlA security.

Security experiment. The challenger proceeds the combined parameter generation of both the PlA and PACA games to fix PINs and attestation key pairs for all tokens. Queries accepted by the challenger include all those defined in the PACA experiment, except for Authorize and Validate, which are replaced by the following queries:

• $\mathsf{Start}(\pi_S^{i,j}, \pi_C^k, T)$. The challenger instructs $\pi_S^{i,j}$ to execute a rchallenge (if $j = 0$) or achallenge (if $j > 0$) to start the registration (for the given token) or authentication (for the $\pi_S^{i,0}$ registration context) for the involved server's true identity id_S and generate a challenge c, which is given to the adversary. We set $\pi_S^{i,j}.c$ as the returned challenge. The challenger also takes $\pi_C^k.\mathsf{bs}$ and uses it to authenticate command M, which is generated by $\mathsf{rcommand}(\mathsf{id}_S, c)$ (if $j = 0$) or $\mathsf{acommand}(\mathsf{id}_S, c)$ (if $j > 0$), and return the result to $\mathcal{A}$. Note that this query captures an authenticated communication between the server oracle and the client oracle: the client oracle is assumed to know the server's true identity id_S and its received challenge c is not tampered.

• $\mathsf{Challenge}(\pi_T^i, \pi_T^{j,k}, (M, t))$. The challenger first takes $\mathsf{st}_T.\mathsf{bs}_i$ and uses it to verify (M, t) based on the user decision sent to π_T^i. If verification is successful, the challenger processes the command using rresponse (if $k = 0$) or aresponse (if $k > 0$) and returns the result R to the adversary using $\pi_T^{j,k}$. We define $\pi_T^{j,k}.R$ as the output R and $\pi_T^{j,0}.\mathsf{rct}$ as the resulting registration context.

• $\mathsf{Complete}(\pi_S^{i,j}, R)$. This query delivers an authenticator response to a server oracle, which performs to process the response using rcheck (if $j = 0$) or acheck (if $j > 0$) and return the result to the adversary. We define $\pi_S^{i,j}.R$ as the input R to the check and $\pi_T^{i,0}.\mathsf{rcs}$ as the resulting registration context.

We do not introduce new notions of partnership. We refer partnership relations between PACA binding session oracles (client and token) as PACA-partnered. We recall that this is defined as having consistent views in the communications during binging. We refer to partnership relations between PlA authentication session oracles (server and token) as PlA-partnered. We recall that this means that the server oracles received the responses sent by the token oracles, and that there is a consistent view of the challenges fixed by the session identifier.

Advantage measure. An adversary $\mathcal{A}$ against a PlA+PACA protocol Π has the following advantage measure called *User Authentication (UA)*. We define $\mathbf{Adv}^{\mathsf{ua}}_{\Pi}(\mathcal{A})$ as the probability that the following does not hold. Take any Complete authentication query accepted by server oracle $\pi_S^{i,j}$. Then, there exist challenges c_r and c_a, authorized commands (M_r, t_r) and (M_a, t_a), gestures G_r and G_a, and oracles $\pi_T^{k,l}$, π_T^m, π_T^n, π_C^x, and $\pi_{C'}^y$, such that:

1. $\pi_S^{i,j}$ and $\pi_T^{k,l}$ are each other's unique PlA partners.
2. $\pi_T^{k,0}$ was created as a consequence of π_T^m accepting command (M_r, t_r) under gesture G_r.
3. $\pi_T^{k,l}$ was created as a consequence of π_T^n accepting command (M_a, t_a) under gesture G_a.
4. π_T^m and π_T^n are unique PACA partners of π_C^x and $\pi_{C'}^y$, respectively.
5. c_r was produced by $\pi_S^{i,0}$; c_r was used by π_C^x as input to generate (M_r, t_r) at a time when π_T^m was valid.
6. c_a was produced by $\pi_S^{i,j}$; c_a was used by $\pi_{C'}^y$ as input to generate (M_a, t_a) at a time when π_T^n was valid.
7. id_S was the server-side input to $\pi_S^{i,0}$ and the client-side input to π_C^x and $\pi_{C'}^y$; The registration contexts of $\pi_S^{i,0}$ and $\pi_T^{k,0}$ both encode id_S.

Note that the server session that accepts authentication fixes unique token-side and client-side sessions involved in both registration and authentication challenge-response protocols. Note also that the server is assured that the same token used in registration is used for authentication due to the

PlA unique partnering property. Finally, there is a notion of freshness in the sense that token sessions were valid when they interacted with the clients, i.e., they were created since the last time the token was powered up.

These advantage measures hold with respect to *weak* channels if the adversary must succeed with the compromise and corruption capabilities of SUF (or SUF-t)[21] PACA attackers. They hold with respect to *strong* channels if the adversary must succeed with the compromise and corruption capabilities of UF (or UF-t) PACA attackers. Note that the first guarantee is strictly stronger, as the adversary is given more power.

Theorem 6 *Take a PACA protocol* Π *that is* suf*-secure and a PlA protocol* Σ *that is* pla*-secure. Then the advantage of any attacker* $\mathcal{A}$ *against the composed protocol* $\Pi+\Sigma$ *with respect to weak channels is bounded by* $\mathbf{Adv}^{\mathsf{ua}}_{\Pi+\Sigma}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{suf}}_{\Pi}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{pla}}_{\Sigma}(\mathcal{C})$ *where* $\mathcal{B}$ *and* $\mathcal{C}$ *are attackers against the PACA and PlA protocols, respectively.*

Proof: [Sketch] Let us now look at the UA requirements, in order:

- [1] We prove the PlA unique partnering property by a direct reduction to the PlA game: we can construct an adversary $\mathcal{C}$ that simulates all the PACA-related side of the experiment and uses the Start, Challenge and Complete oracles in the PlA game to simulate the corresponding queries in the composed security experiment. Note that, since no restriction is placed on the adversarial queries in the PlA experiment, if the uniqueness condition of PlA partners is violated in the simulation, then this implies that the same property is violated in the PlA game.

- [2,3,4] First note that the unique partnering property proved above guarantees that a registration command and an authentication command must have been accepted by the token holding the PlA partner to create the PlA sessions, as otherwise the token would not have produced the corresponding responses. This shows that commands (M_r, t_r) and (M_a, t_a) must exist. Furthermore, PACA security guarantees that command acceptance implies the existence of unique (valid) client-to-token partnering and accepting user gestures. We can therefore construct an adversary $\mathcal{B}$ that that breaks PACA security whenever such client sessions or gestures do not exist.

- [5,6] These points state claims that the client oracles π^x_C and $\pi^y_{C'}$ must in fact have received the single challenges produced by $\pi^{i,0}_S$ and $\pi^{i,j}_S$, respectively, in order to generate (M_r, t_r) and (M_a, t_a). This property follows from PlA security, as otherwise it would contradict unique partnering. This is therefore handled by $\mathcal{C}$ as a special case.

- [7,8] The properties proved above guarantee that the commands accepted by the token were correctly created according to the PlA protocol: we have server-to-client and client-to-token authenticity). Furthermore, the PlA partnership guarantees response equality, and hence we can appeal to PlA correctness and the conditions on the inputs: the server would not have accepted unless there is an exact match on inputs, which then implies the conditions on registration context outputs.

□

This theorem does not directly apply to the WebAuthn+CTAP2 combination because CTAP2 does not meet SUF security. However, it is easy to see that a weakening of the theorem holds, where one restricts the adversary's abilities in the same way as in the PACA UF-t definition (i.e., with respect to strong channels and without active attacks against a PACA client).

[21] Note that the compromise and corruption capabilities of SUF and SUF-t are the same, likewise for UF and UF-t.