Provable Security Analysis of FIDO2

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

[1]Georgia Institute of Technology
shanchen@gatech.edu sasha@gatech.edu
[2]University of Porto
mbb@fc.up.pt
[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 new Client-to-Authenticator Protocol (CTAP2) and the Web Authentication (WebAuthn) specification.

Our analysis is *modular*. For CTAP2 and WebAuthn, in turn, we propose appropriate security models that aim to capture their intended security goals and use the models to analyze security. We identify a series of shortcomings and propose stronger protocols designed to withstand stronger yet realistic adversaries. Next, we prove the security guarantees FIDO2 provides 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, our proposed constructions should pave the way towards the design and deployment of more secure passwordless user authentication protocols.

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 [17], 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 [31]. 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 [28].

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, 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: Client-to-Authenticator Protocol v2.0 (CTAP2[1]) and W3C's Web Authentication (WebAuthn).

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 [19]. To understand the exact security guarantees and fix any potential vulnerabilities before wide deployment of the FIDO protocols, it is critical and urgent to provide their formal security analyses.

Related Work. Some initial work in this direction already exists. Hu and Zhang [22] analyzed the security of FIDO UAF 1.0 and identified several vulnerabilities in different attack scenarios. Later, Panos *et al.* [29] 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 [20,24,30]. Regarding an older version of FIDO U2F, Pereira *et al.* [30] presented a server-in-the-middle attack and Jacomme and Kremer [24] further analyzed it with a structured and fine-grained threat model for malware. Guirat and Halpin [20] 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 used among different servers.

Note however that *none* of the existing work employs the provable security approach for the FIDO2 protocols in the course of deployment. In particular, there is no analysis of CTAP2, and the results for WebAuthn [24] are quite insufficient. 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". Moreover, their analysis uses the *symbolic* model (often called the Dolev-Yao model [18]), which captures weaker adversarial capabilities than those in computational models (e.g., the Bellare-Rogaway model [11]) employed by the provable security approach.

The works on two-factor authentication (e.g., [16, 26]) are related, but in such protocols the user has to use the password *and* the second-factor device during each authentication/login. With FIDO2, there is no password use during the user registration or authentication phases. Passwords (PINs) only occur in the initialization phase which "registers" the security device to a browser rather than a server. 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 prevents 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 for the token. There are no full security models defined for federated authentication, but they should be quite different. It is interesting to see how FIDO2 and federated authentication can be used securely together; we leave it for future work. Our model could 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 to help clarify the formal trust model and its impact on the 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.

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

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

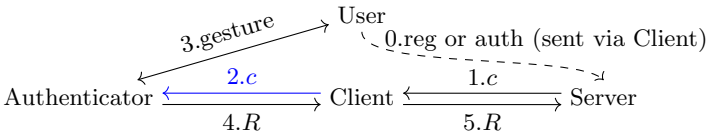

Figure 1: FIDO2 flow (simplified): blue = CTAP2 authenticated message.

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 an authenticator device, which is a user-owned security device (e.g., a security token or a smartphone). Such a device-assisted "challenge-response" scheme works as follows (details in Section 8). First, in the registration phase, the server sends a random challenge to the security device through a client (e.g., a browser or an operating system installed on the user's laptop). In this phase, the device signs the challenge using its long-term embedded 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 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 a client (such as a browser) that has a user PIN as input. 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 CTAP2 relies on the (unauthenticated) Diffie-Hellman key exchange. The details can be found in Section 5.

Our Contributions. We perform the first thorough cryptographic analysis of the authentication properties guaranteed by FIDO2 protocols using the provable security approach: first defining the protocol syntax, then designing an appropriate security model that specifies the adversarial capabilities and formal security goals, and finally proving security by reduction to the security of the building blocks or identifying attacks captured by the model. To better understand and improve each component protocol, our analysis is conducted in a *modular* way. That is, we analyze CTAP2 and WebAuthn separately and then derive the overall security of a typical use of the FIDO2 protocols.

Provable security of CTAP2. We start our analysis with 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 party and therefore captures human interactions with the client and authenticator (e.g., the user typing its PIN into the client 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 the client and as a result the authenticator stores a PIN-related long-term state that we call a transformed PIN. Then, in the binding phase, the client (with the same input PIN) is "bound" to the authenticator. At the end of this phase, each party ends up with a (perhaps different) binding state. Finally, in the authenticated channel phase, the client is able to send any authenticated message (computed using its binding state) to the authenticator, which verifies it using its own binding state. Note that the final established authenticated channel is *unidirectional*, i.e., it only guarantees authenticated access from the client to the authenticator but not the other way.

The end-goal of CTAP2 is to grant a client exclusive access to the user's authenticator device, by establishing an authenticated channel from the client to the authenticator. Our model captures the security of the authenticated 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 power-up) to the trusted client and the pinToken is

then stored on the client as the binding state. Later in the authenticated 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 authenticated channels with the same authenticator using the *same* pinToken. This limits the security of CTAP2 authenticated channels: for a particular channel from a client to an authenticator 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 channels for which we define security. As we explain later (Section 4), 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 6. Based on our security model, we prove that CTAP2*, a simplified version of CTAP2 that removes a redundant protocol message, achieves the weakest UF-t security. We also show that CTAP2 is not secure regarding the three stronger notions.

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 the 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 authenticated 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 argue that sPACA comes with only little overhead over CTAP2 and requires only minimal changes, so should be considered for adoption.

Provable security of WebAuthn. Next, we define the class of *Passwordless Authentication (PlA)* protocols which aim 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) and 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 bound to the authenticator's attestation public key vk, a username id_U, and a server name id_S. 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 $(vk, \mathsf{id}_U, \mathsf{id}_S)$.

Technically, we formalize the intuition outlined above using the concept of *partnered sessions*. Two sessions (one on the authenticator side and one on the server side) are partnered if it is clear that they communicate with one another. We borrow one formalization of this intuition from the key-exchange literature, and demand that protocol sessions locally derive session identifiers which should somehow capture the intended partner. For challenge-response protocols (as the ones we analyze in this paper), the session identifier can be defined simply, as the challenge signed by the authenticator device.

Roughly speaking, a PlA protocol is secure if, whenever a registration session completes on the server side, there is a unique partnered registration session which completed successfully on the authenticator side. For authentication sessions we further impose that the associated registration sessions (as defined independently on both sides) are also uniquely partnered. This guarantees that registration contexts 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, and must have sent the reply which the server accepted in the authentication session. 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 8.

Composed security of CTAP2* and WebAuthn. Finally, towards the analysis of full FIDO2 (by full FIDO2 we mean the envisioned usage of the two protocols), we study the composition of PACA and PlA protocols (cf. Section 9). The composed protocol, which we simply call PACA+PlA, 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 authenticated 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 and prove an even stronger property that shows that PACA+PlA gives end-to-end mutual authentication guarantees between the server and the authenticator when clients and servers are connected by a server-to-client authenticated channel. The mutual authentication guarantees extend the PlA guarantees: authenticator, client, and server must all be using the same registration context $(vk, \mathsf{id}_U, \mathsf{id}_S)$ 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 CTAP2*+WebAuthn (under a UF-t adversarial model) and sPACA+WebAuthn (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 usernames and server names are being used in PlA sessions.

We hope our analyses will help clarify the security guarantees of the FIDO2 protocols and expect our proposed constructions to facilitate the design and deployment of more secure passwordless user authentication protocols.

2 Preliminaries

Notations. Let $\{0,1\}^*$ denote the set of all finite-length binary strings (including the empty string ε) and $\{0,1\}^n$ denote the set of n-bit binary strings; for a binary string s, let $|s|$ denote its length in

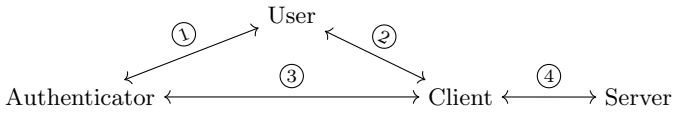

Figure 2: Communication channels

bits. We use $[n]$ for the set of integers $\{1, 2, \ldots, n\}$ and use the wildcard $\cdot$ to indicate any valid input of a function or algorithm. For a finite set $\mathcal{R}$, let $|\mathcal{R}|$ denote its size and $r \xleftarrow{\$} \mathcal{R}$ denote sampling r uniformly at random from $\mathcal{R}$. We use $y \leftarrow x$ for assigning a value to a variable; in particular, if f is a deterministic function, we use $y \leftarrow f(x)$ to denote 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.

In Appendix A, we recall the definitions of pseudorandom functions, message authentication codes, signature schemes, collision-resistant hash function families, the computational Diffie-Hellman (CDH) problem and strong CDH problem, as well as the corresponding advantage measures $\mathbf{Adv}^{\mathsf{prf}}_{F}$, $\mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{MAC}}$, $\mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{Sig}}$, $\mathbf{Adv}^{\mathsf{coll}}_{H}$, $\mathbf{Adv}^{\mathsf{cdh}}_{\mathbb{G},g}$, $\mathbf{Adv}^{\mathsf{scdh}}_{\mathbb{G},g}$. In Appendix B, we recall the syntax for PAKE and its security including 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) $\mathcal{T}$, 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 7.

The possible communication channels are represented as double-headed arrows in Figure 2. In CTAP2+WebAuthn, 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 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.* [13], we require that messages sent to the user be *human-readable* and those sent by the user be *human-writable*.[3] 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 power-down/power-up cycle, which we call a *reboot*. Static storage is assumed to be initialized using a procedure carried out under special set-up trust assumptions; in the case of this paper we will consider the set-up procedures to configure a PIN, i.e., "embedding" the user PIN in the authenticator, and generating an attestation key pair for the authenticator.

[3] We regard understandable information from Internet browsing as human-readable and typing in a PIN or rebooting an authenticator as human-writable.

Trust model. For each of the protocols we analyze in the paper we specify a trust model, which justifies the modeling of the security experiments. Here we state the trust assumptions that are always assumed throughout the paper. Human communications (①②) 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. Client-authenticator communications (③) are not protected, i.e., neither authenticated nor private. The authenticator is assumed to be tamper-proof, so the model will not consider corruption of its 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 name 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. Indeed, 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 authenticator 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 $d = G(x, y)$, 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 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 $G(\mathsf{id}^*, \mathsf{id}) := (\mathsf{id}^* \stackrel{?}{=} \mathsf{id})$.

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

4 PIN-Based Access Control for Authenticators

In this section, in order to study FIDO2's CTAP2, 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 setup and possibly an arbitrary number of authenticator reboots, the user can use the client to issue PIN-authenticated commands to the token, which the token can use for access control, e.g., to unlock built-in functionalities that answer client commands.

4.1 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 authenticator T, denoted by st_T, is partitioned into the following components: i) static storage $\mathsf{st}_T.\mathsf{ss}$; ii) power-up or reset state $\mathsf{st}_T.\mathsf{rs}$; and iii) one or more binding states $\mathsf{st}_T.\mathsf{bs}_i$ (together denoted by $\mathsf{st}_T.\mathsf{bs}$). A client C may also have multiple binding states, which we denote by $\mathsf{bs}_{C,j}$.

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

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 $\mathsf{st.ss}$ 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.

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

Bind This subprotocol is executed by the three 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 power-up state. At the end of this phase, in the case of success, both the token and the client get a new binding state; the token may update its power-up state (e.g., a counter).[4] If the subprotocol fails, no binding states are set, but the token power-up state may still be updated. We assume the client always initiates this subprotocol once it gets the PIN from the user.

Authenticate This algorithm allows a client to generate authenticated commands for the authenticator. The client inputs a command M and a binding state bs_j. We will use $(M,t) \xleftarrow{\$} \mathsf{authenticate}(M, \mathsf{bs}_j)$ to denote the generation of an authenticated command.

Verify This algorithm allows a token to verify authenticated commands sent by a client with respect to a binding state and a user decision. The token gets a public predicate G that captures the user's decision semantics; $G(x,y)$ models a decision based on information x, y respectively conveyed by the client and the token, with y depending on M and $\mathsf{st.bs}_i$ that the token may display to the user directly.[5] The token inputs an authenticated command (M,t), a binding state $\mathsf{st.bs}_i$, and a user decision $d = G(x,y)$. We denote $\mathsf{verify}((M,t), \mathsf{st.bs}_i, d)$ as the deterministic computation performed by the token to obtain an accept or reject indication.

Correctness. Correctness is defined for an arbitrary public predicate G. We consider any token T, and any sequence of commands of the following form: i) a successful setup using PIN fixing $\mathsf{st}_T\mathsf{.ss}$ via some client; ii) any sequence of subprotocols excluding setup via arbitrary clients; iii) a binding with PIN creating client-side binding state $\mathsf{bs}_{C,j}$ at a client C and token-side binding state $\mathsf{st}_T\mathsf{.bs}_i$; iv) any sequence of subprotocols executed via C, excluding setup and reboot; v) authentication of command M by C as $(M,t) \xleftarrow{\$} \mathsf{authenticate}(M, \mathsf{bs}_{C,j})$; and vi) verification by the token as $\mathsf{verify}((M,t), \mathsf{st}_T\mathsf{.bs}_i, d)$. Correctness requires that verification is successful iff $G(x,y) = 1$ (i.e., $d = 1$) holds. A correct PACA will further impose that $\mathsf{st}_T\mathsf{.ss}$ is unchanged and that $\mathsf{st}_T\mathsf{.bs} = \perp$ after a reboot.

4.2 Security Model

Trust model. Before defining our security model, we first state the assumed security properties for the communication channels shown in Figure 2. The only restriction is that the Setup subprotocol is assumed to be carried out over a communication channel where the adversary can only eavesdrop communications between the client and authenticator; this is a necessary condition, as there is 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 one another and they are assumed to be available throughout the experiment execution. For tokens, the static storage and power-up states are maintained by the security experiment; session oracles capture binding states. Binding states at the

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

[5] For example, G may encode a partnering code or a session identifier that allows the user to confirm the command came from a specific browser window; it can also be the trivial predicate that always returns true, meaning that the user cannot actually check any details of the command by physically interacting with the token.

token become *invalid* after rebooting if they were not empty at the time of the reboot, which means that the adversary is thereafter blocked access to such oracles. However, we keep these oracles in the game in order to capture strong authentication guarantees across reboots.

Security experiment. The security experiment is run between a challenger and an adversary $\mathcal{A}$. In the beginning of the experiment, the challenger resets states of all tokens, i.e., it sets $\mathsf{st}_T \leftarrow \perp$ for all $T \in \mathcal{T}$. It then samples independent random PINs for all tokens, i.e., $\mathsf{pin}_T \xleftarrow{\$} \mathcal{PIN}$ for all $T \in \mathcal{T}$; note that only one PIN is needed per token, since we assume a single setup. The adversary $\mathcal{A}$ interacts with the challenger via the following queries:

• $\mathsf{Setup}(\pi_T^i, \pi_C^j)$. The challenger runs the Setup subprotocol between π_T^i and π_C^j using pin_T. It returns the trace of communications between π_T^i and π_C^j to $\mathcal{A}$. $\mathsf{st}_T.\mathsf{ss}$ is set for the rest of the experiment. Oracles created for setup must never have been used before and are always declared *invalid* after setup completion.[6]

• $\mathsf{Reboot}(T)$. The challenger executes the Reboot algorithm for token T, marking all previously used instances π_T^i as *invalid* and setting $\mathsf{st}_T \xleftarrow{\$} \mathsf{reboot}(\mathsf{st}_T.\mathsf{ss})$.

• $\mathsf{Execute}(\pi_T^i, \pi_C^j)$. The challenger runs the Bind subprotocol between π_T^i and π_C^j using pin_T. It returns to $\mathcal{A}$ the trace of communications between π_T^i and π_C^j. This query allows the adversary to access honest protocol executions in which it cannot take active action to guess the user's PIN. The resulting binding states on both sides are kept as $\mathsf{st}_T.\mathsf{bs}_i$ and $\pi_C^j.\mathsf{bs}$ respectively.

• $\mathsf{Connect}(T, \pi_C^j)$. The challenger instructs π_C^j to initiate the Bind subprotocol with T using pin_T. It returns the first message sent by π_C^j to $\mathcal{A}$. Note that no client-side oracles can be created if Connect queries are disallowed, since we assume the client is the initiator of the Bind subprotocol. This query allows launching an active attack against a client-side 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 $\pi_C^i.\mathsf{bs}$ for a client oracle. This query allows the adversary to launch an active attack against a token-side oracle or completing an active attack against a client-side oracle.

• $\mathsf{Authenticate}(\pi_C^i, M)$. The challenger takes $\pi_C^i.\mathsf{bs}$ and uses it to authenticate command M and return the result to $\mathcal{A}$.

• $\mathsf{Verify}(\pi_T^i, (M, t))$. The challenger takes $\mathsf{st}_T.\mathsf{bs}_i$ and uses it to verify (M, t) based on the user decision sent to π_T^i. The result is returned to $\mathcal{A}$.

• $\mathsf{Compromise}(\pi_C^i)$. This query returns $\pi_C^i.\mathsf{bs}$. After this query, we say π_C^i was *compromised*.

• $\mathsf{Corrupt}(T)$. This query returns pin_T. After this query, we say T was *corrupted*. All queries are ignored if they refer to any oracle π_P^i that has been marked invalid.

Partners. We say an authenticator oracle π_T^i and a client oracle π_C^j are each other's *partner* if they have both completed their binding phases and their views are consistent. We take these views as a *session identifier* sid that must be defined for each protocol and must be the same for both parties, usually as a function of the communications trace. 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 an authenticator is rebooted then its power-up state is refreshed and all of its binding states (if any) are invalidated. Partnering is defined over all, even invalid binding states. Intuitively we will require that access control is associated with a unique *valid* partner, i.e., the client knows that an issued command will not be accepted if the authenticator was rebooted before command delivery.

Security goals. We define four levels of security for PACA. All advantage notions define PAKE-like security where the adversary's probability of success should be negligibly larger than the trivial attack of guessing the PIN using an online attack launched with Connect and Send queries. In our theorems we fix the number of queries to Connect as an adversarial parameter while (unlike for PAKE) we do

[6] Corrupting π_C^j would imply revealing the PIN, so we do not allow this. However, these oracles can still influence the partnering relation and hence PACA security implies that session identifiers for setup sessions must not create ambiguity with those of binding sessions. This restriction implies that whatever client sessions are used for setup, no state about them remains when the token is subsequently used.

not do this for Send. Instead, we require PACA protocols to include token-side blocking counter to limit the total number of failed PIN guessing attempts (across reboots).

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

1. Gesture G approves M, i.e., $G(x, y) = 1$;
2. (M, t) was output by one of T's partners π^j_C;
3. π^i_T and π^j_C have unique *valid* partners.

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

The above captures the attacks in which the attacker successfully makes an authenticator accept a forged authenticated command, without corrupting the user who set up the authenticator or compromising any of the authenticator's partners. A PACA protocol satisfying the above security notion prevents an attacker from sending commands to the authenticator even if it stole the authenticator, unless the attacker corrupts the user PIN that was used to set up the authenticator or compromises any of the authenticator's partners. Note that the authenticated channels considered in this notion have only weak security, i.e., compromising one channel implies compromising all channels (to the same authenticator and since the last reboot). The requirement that the involved session oracles have unique valid partners guarantees a binding between an accepted commands and a unique binding session in the entire lifetime of the authenticator *and* this unique session was created since the last reboot.

Unforgeability with trusted binding (UF-t). We define its advantage measure $\mathbf{Adv}^{\mathsf{uf\text{-}t}}_{\Pi}(\mathcal{A})$ the same as $\mathbf{Adv}^{\mathsf{uf}}_{\Pi}(\mathcal{A})$ except that the adversary is *not* allowed to make Connect queries. Note that unlike in the trusted setup, here in the binding subprotocol the adversary is still allowed to interact with the authenticator or the client (via Send queries) but it cannot create any pending/interrupted client sessions that allow it to launch a man-in-the middle attack. It can still, however, perform an online dictionary attack on the authenticator. This restriction captures the minimum requirement for proving the FIDO2's CTAP2 protocol using our model, and this is the only reason we define it. Clearly, UF security implies UF-t security.

Strong unforgeability (SUF). We define $\mathbf{Adv}^{\mathsf{suf}}_{\Pi}(\mathcal{A})$ as the UF advantage with a single difference. The adversary is allowed to win the game even if it has compromised any of T's partners, provided that it has not compromised π^j_C, the client bind instance that has a unique partner in π^i_T. Furthermore, the adversary is now allowed to corrupt the PIN for the token before the command is accepted, provided the binding subprotocol was already complete.

The above captures similar attacks considered in UF but in the strong sense, where the adversary is further allowed to compromise some of the target authenticator's partners (except the partner in the target authenticated channel) and corrupt the user even before the forged command was accepted (but after the authenticator set its binding state). The latter relaxation guarantees *forward secrecy* for authentication, which is not as strong as forward secrecy for confidentiality because breaking forward secrecy for authentication does not affect already authenticated commands but only affects future commands sent through an already established authenticated channel. Nevertheless, forward secrecy is still preferable. Besides, unlike UF, authenticated channels considered in this notion have strong security, i.e., compromising one channel does not affect other channels. Obviously, SUF security implies UF security.

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

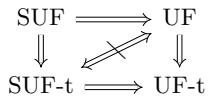

Figure 3: Relations between PACA security notions.

Relations between security notions. As discussed above, Figure 3 shows the implication relations among our four defined notions. UF and SUF-t do not imply each other and we will give examples of separations in Sections 5 and 6.

Improving PACA (S)UF-t security with test of user presence. Under the trusted binding assumption, we can make a simple modification to PACA protocols to eliminate online password guessing attacks altogether. Trusted binding excludes active attacks against the client and, in particular, man-in-the-middle attacks. Nevertheless, there is still possibility of an online dictionary attack against the authenticator. To avoid this, one only needs to test the user presence (e.g., requiring a gesture) at the beginning of the binding phase. We argue that such test-of-user-presence overhead is quite small for CTAP2-like protocols because the user has to type his PIN into the client anyway. The security gain is considerable, because now *no* malicious binding attempts can happen, which opens the way for a neat negligible security bound.

5 The Client to Authenticator Protocol v2.0

We now define the FIDO Alliance's Client to Authenticator Protocol v2.0 (CTAP2) [1] using PACA syntax and present its security analysis.

Protocol Description. CTAP2 consists of several subprotocols (requested with the corresponding sub-commands). For our analysis we focus on its core subprotocols (see Figure 4): $\mathsf{getKeyAgreement}$, setPIN, $\mathsf{getPINToken}$, and $\mathsf{usePINToken}$, [7] (cf. Figure 1 in [1]). where in the beginning the client inputs the user PIN pin_U. In the figure we describe the protocols close to how CTAP2 specs do and later explain how they match PACA syntax.

The PIN space $\mathcal{PIN}$ consists of 64-byte[8] strings. ECKG denotes the key generation function of elliptic-curve Diffie-Hellman (ECDH) that samples an elliptic-curve public and secret key pair (aG, a), where G is an elliptic-curve point generating a cyclic group $\mathbb{G}$ of prime order Q and a is a random number in $\{1, \ldots, Q-1\}$. Note that CTAP2 uses the P-256 elliptic-curve parameter set [23] for 128-bit security. H denotes the SHA-256 hash function (with output truncated to the first 128 bits). $\mathsf{CBC}_0 = (\mathcal{K}, \mathsf{E}, \mathsf{D})$ denotes the (deterministic) AES256-CBC encryption scheme with IV = 0 and $\mathsf{HMAC} : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^{\mathsf{bl}}$ denotes the HMAC-SHA256 message authentication code (with output truncated to the first block size $\mathsf{bl} = 128$ bits).

Let us see how CTAP2 fits into PACA syntax:

- The Setup subprotocol consists of $\mathsf{getKeyAgreement}$ followed by setPIN. The static storage space is occupied with a *transformed PIN*, which is the hash result of the user PIN. A parameter n_{max} is also kept to manage the number of allowed unsuccessful executions of the Bind subprotocol.
- The Reboot algorithm reboot is defined as follows (where pt denotes the random pinToken); the power-up state includes ECDH public and secret parameters, pt, and a counter inc.[9]

[7] Our analysis ignores $\mathsf{getRetries}$ and $\mathsf{changePIN}$. $\mathsf{getRetries}$ is used by the client platform to retrieve the retries counter ctr stored in the token. If ctr is close to 0 (which means the token is easy to get locked), the client may warn the user to be careful while entering the PIN. This information is public in our model. $\mathsf{changePIN}$ does involve some security properties, but it is equivalent to running $\mathsf{getPINToken} + \mathsf{setPIN}$. We ignore this aspect to reduce the complexity of our analysis and presentation since it does not detract from the main take-aways.

[8] PINs memorized by users are of length 4∼63 bytes in UTF-8 representation, padded with trailing 0 bytes.

[9] Note that it completely ignores static storage, i.e., it is PIN-independent.

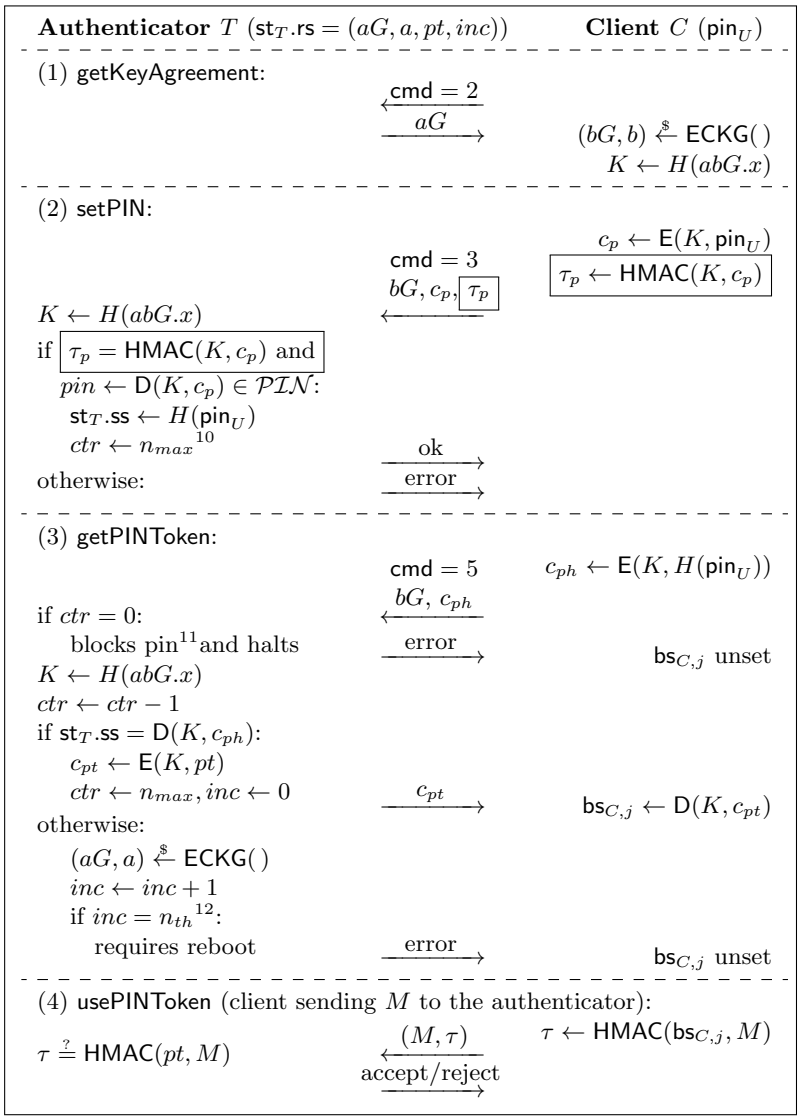

Figure 4: The CTAP2 protocol (and CTAP2* that excludes the boxed contents).

reboot(st.ss):
$(aG, a) \xleftarrow{\$} \mathsf{ECKG}()$, $pt \xleftarrow{\$} \{0,1\}^{\mathsf{bl}}$, $inc \leftarrow 0$
return $(\mathsf{st.ss}, \mathsf{st.rs} \leftarrow (aG, a, pt, inc), \mathsf{st.bs} \leftarrow \perp)$

- The Bind subprotocol consists of getKeyAgreement followed by getPINToken. The binding states are simply pt.
- The authenticate and verify algorithms are given by the client-side and authenticator-side computations in usePINToken. Note that, although the token-side computations do not explicitly refer to a gesture G, it is implicit that the authenticator may display information about M and pt to the user and ask for confirmation before approving the command. This would allow the user to cross-check with the browser window to confirm a legitimately issued command.

We omit a proof that CTAP2 is a correct PACA protocol, but this is straightforward to check from the figure. Furthermore, in our security analysis we assume that there exists some class of gestures that tokens can efficiently compute and that any authenticated command validated by the authenticator is only accepted if a gesture is given to approve it. The semantics of gestures is relevant for the client-side guarantees we discuss in Section 9. The session identifier is defined as the full communications trace in the binding phase. (Note that such a trace can never match a trace in the authenticator setup phase.)

Security Results. First, we notice that the MAC authentication (boxed in Figure 4) in setPIN is useless, i.e., it does not provide any authentication protection for a MITM attacker. Such an attacker can pick its own ECDH key share to compute the shared key K that is used to generate valid ciphertexts and authentication tags. However, using the same key K for both encryption and authentication is considered bad practice and the resulting security guarantee is elusive for the CBC construction.[13] Therefore, in our security analysis, we remove those redundant and problematic MAC operations and focus on the resulting simplified protocol (denoted by CTAP2*) which is more efficient and at least as secure as the original CTAP2.

Insecurity of CTAP2. It is not hard to see that CTAP2 is neither UF secure nor SUF-t secure (and hence SUF insecure). If the binding phase is not trusted, an attacker can impersonate the authenticator to get the PIN hash (i.e., transformed PIN) from the client which takes the user PIN as input. Then, the attacker is able to get pt from the authenticator. On the other hand, if any of the authenticator's partners is compromised, the attacker is able to get pt shared between them.

UF-t security of CTAP2*. Note that in practical scenarios, if an authenticator is stolen by the attacker, n_{max} limits the maximum number of consecutive wrong PIN guesses before the authenticator blocks further interactions. On the other hand, if the target authenticator is not stolen (i.e., possessed by a user), then authenticator reboots imply user detectability. Therefore, n_{th} ($< n_{max}$) limits the maximum number of *undetectable* consecutive wrong PIN guesses after each honest binding execution. The following theorem (proved in Appendix C) confirms UF-t security of CTAP2*, when modeling the hash function H as a random oracle $\mathcal{H}$.

Theorem 1 *Let $\mathscr{D}_{\mathcal{PIN}}$ be an arbitrary distribution over the PIN dictionary $\mathcal{PIN}$ with min-entropy $h_{\mathscr{D}}$.*[14] *For any efficient adversary $\mathcal{A}$ making at most q_{S} queries to* Setup*, q_{E} queries to* Execute*, q_{R} queries to* Reboot*, q_{A} queries to* Authenticate *and $q_{\mathcal{H}}$ queries to $\mathcal{H}$ in the random oracle model, there exist efficient adversaries $\mathcal{B}, \mathcal{C}, \mathcal{D}$ such that:*

$$\begin{aligned}\mathbf{Adv}^{\mathsf{uf\text{-}t}}_{\mathrm{CTAP2}^*}(\mathcal{A}) \leq\ & nq_{\mathsf{S}}/2^{h_{\mathscr{D}}} + (q_{\mathsf{S}} + q_{\mathsf{E}}) \cdot \mathbf{Adv}^{\mathsf{scdh}}_{\mathbb{G},g}(\mathcal{B}) + 2(q_{\mathsf{S}} + q_{\mathsf{E}}) \cdot \mathbf{Adv}^{\mathsf{prf}}_{AES\text{-}256}(\mathcal{C}) \\ & + q_{\mathsf{S}} q_{\mathsf{R}} \cdot \mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{HMAC}}(\mathcal{D}) + ((n+12)q_{\mathsf{S}} + (q_{\mathsf{A}} + q_{\mathcal{H}})^2) \cdot 2^{-\mathsf{bl}} + {q_{\mathsf{E}}}^2/Q,\end{aligned}$$

where $n = n_{th}$ if user undetectability is required for $\mathcal{A}$ or $n = n_{max}$ otherwise.

To see that the above guarantee applies to the CTAP2* instantiation it remains to note that AES-256 is believed to be a PRF and HMAC-SHA256 has been proved to be a PRF (and hence EUF-CMA) [8] assuming SHA256's compression function is a PRF.

SUF-t does not imply UF. Note also that CTAP2* becomes SUF-t secure (but still not UF secure) if an independent pinToken is used for each binding session. This shows that SUF-t security does not imply UF security.

Avoiding authenticator reboots caused by consecutive PIN mismatches. As mentioned above, the n_{th} threshold is used for user detectability. To involve user interaction, CTAP2 requires the

[10] By default $n_{max} = 8$.

[11] Once the authenticator blocks the pin, it needs to be reset to the factory default state (i.e., erasing all previous state) before further operations.

[12] By default $n_{th} = 3$.

[13] Our recommendation is that these should be dropped or replaced with non-authenticated checksums.

[14] A user-memorizable PIN has low entropy, so $h_{\mathscr{D}} < \log|\mathcal{PIN}| = 512$.

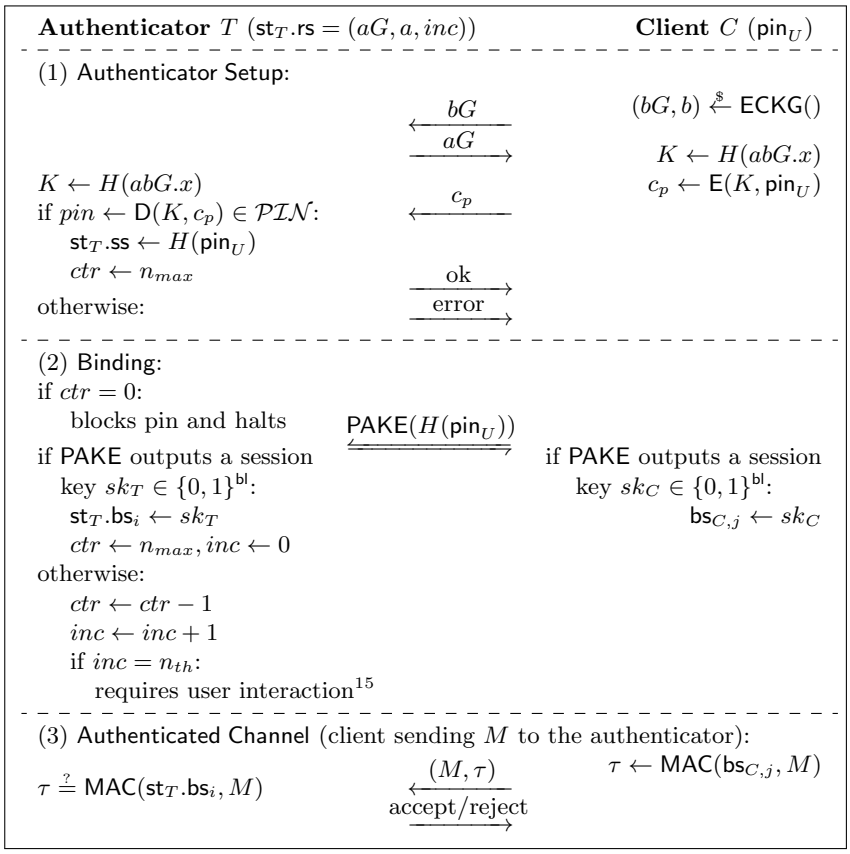

Figure 5: The sPACA protocol

authenticator to reboot each time n_{th} consecutive PIN mismatches occur. Such reboots do not enhance security because the stored transformed PIN is not updated, but they could cause usability issue because each reboot invalidates all existing client-authenticator bindings. Therefore, we recommend it instead require a simple test of user presence (e.g., pressing a button) as well as resetting the *inc* counter to 0, when n_{th} consecutive PIN mismatches are detected.

6 Fully Secure PACA Protocol

In this section, we propose a (minimally) modified version of CTAP2 and prove its SUF security. We call it sPACA for secure PACA.

sPACA consists of three subprotocols (as shown in Figure 5) that respectively correspond to the three PACA phases. It employs the same cryptographic primitives as CTAP2, as well as a PAKE protocol. Intuitively, the PAKE replaces the binding subprotocol and uses a hash of the PIN as the password. This means that CTAP2's ECDH protocol is only used for PIN setup. Its authenticator power-up state generation function reboot just sets counter *inc* to 0.

SUF security of sPACA. The following theorem shows the SUF security of sPACA. The session identifier for binding sessions is simply that defined by the underlying PAKE. The proof is in Appendix D:

[15]The user interaction could be pressing a button (which resets $inc = 0$).

Theorem 2 *Let PAKE be a 3-pass protocol where the client is the initiator and let $\mathscr{D}_{\mathcal{PIN}}$ be an arbitrary distribution over the PIN dictionary $\mathcal{PIN}$. For any efficient adversary $\mathcal{A}$ making at most q_{S} queries to* Setup*, q_{C} queries to* Connect*, q_{E} queries to* Execute*, and $q_{\mathcal{H}}$ queries to $\mathcal{H}$ in the random oracle model, there exist efficient adversaries $\mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}$ such that:*

$$\mathbf{Adv}^{\mathsf{suf}}_{\mathrm{sPACA}}(\mathcal{A}) \leq q_{\mathsf{S}}\mathbf{Adv}^{\mathsf{cdh}}_{\mathbb{G},g}(\mathcal{B}) + 2q_{\mathsf{S}}\mathbf{Adv}^{\mathsf{prf}}_{AES\text{-}256}(\mathcal{C}) + \mathbf{Adv}_{\mathsf{PAKE}}(\mathcal{D}, 2(nq_{\mathsf{S}} + q_{\mathsf{C}}), \mathscr{D}_{\mathcal{PIN}}) + (q_{\mathsf{C}} + q_{\mathsf{E}}) \cdot \mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{MAC}}(\mathcal{E}) + (12q_{\mathsf{S}} + 2{q_{\mathcal{H}}}^2) \cdot 2^{-\mathsf{bl}},$$

where $n = n_{th}$ if user undetectability is required for $\mathcal{A}$ or $n = n_{max}$ otherwise.

Note that it is crucial for PAKE to guarantee *explicit* authentication, otherwise, the authenticator might not be able to detect wrong PIN guesses and decrease its *ctr* counter used to prevent exhaustive PIN guesses.[16] Also note that the PAKE advantage bound may itself include calls to an independent random oracle. Such a PAKE can be instantiated with variants of CPACE [21] 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.[17] 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, 21].

UF does not imply SUF-t. Note that one can easily transform sPACA into a protocol that is still UF secure, but not SUF-t secure: let the authenticator generate a global pinToken used for authentication as with CTAP2 and send it (encrypted with the session key output by PAKE) to its partner in the end of the binding phase. This shows that UF security does not imply SUF-t security.

Performance comparison of CTAP2 and sPACA. We purposely design our sPACA protocol following CTAP2 such that the required modification is minimized. To achieve stronger security, sPACA introduces slightly more overhead for binding (in terms of group exponentiations, hashes, AES computations, and communication complexity), as summarized in Table 1. There sPACA is instantiated with CPACE, which requires one and a half round trips or three flows (while CTAP2 only runs one round trip) when explicit authentication is required; the last client-to-token message provides client authentication. However, if the binding is performed when the client already has a command to issue, then this last PAKE message can be piggy-backed with the authenticated command, leading to essentially no overhead.[18] For the proposed instantiations, the token-side computational cost is that of plain Diffie-Hellman if tamper-proof storage can be assumed; this is because both protocols allow precomputing a password-dependent component that removes the online overhead.[19] In short, the only additional computational cost for the authenticator over CTAP2 is computing and verifying the PAKE key confirmation messages (which leads to 3 more hashes), but CPACE does not involve AES computations. (Note that the most expensive computation cost is group exponentiation, for which both protocols have two.) We also 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 the authenticator setup phase to achieve the same SUF security.

A practical implication of SUF security. We note that SUF security offered by sPACA has a practical meaning: commands can be traced back to a unique binding session identifier, as corrupting one client session does not allow forging commands for another session. This means that an authenticator that allows a user to confirm a session identifier for a command can allow a human user to

[16] One does not actually need explicit token-to-client authentication in the proof, as the client does not have long-term secret to protect. This would potentially 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.

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

[18] 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 password guessing attacks against the token as stated in Section 4.

[19] Usually this precomputation cannot be done because corrupting the state of a party would allow an offline dictionary attack.

Table 1: Performance comparison of CTAP2 and sPACA for binding

Protocol	Flow	Authenticator			Client			Comm.
		exp	H	E	exp	H	E	(bl)
CTAP2	2	2	1	2	2	1	2	6
sPACA	3	2	4	0	2	4	0	6

detect rogue commands issued by an adversary (e.g., malware) that corrupted one of the clients (e.g., browser window) bound to the token.

7 Passwordless Authentication

In this section, we define the syntax and security model for *Passwordless Authentication (PlA)* protocols. This is in order to analyze the security of FIDO2's WebAuthn protocol.

7.1 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 syntax of the protocol also permits binding the authentication parameters to a user-server name pair.

The possible communication channels are represented as double-headed arrows in Figure 2. The model overlaps with that for PACA protocols in that we consider client-to-authenticator communications, but we do not include the user, and consider servers as a new type of protocol entity. Servers are accessible to clients via a communication channel that models Internet communications.

Syntax. The set of parties $\mathcal{P}$ consists of three disjoint sets of parties: authenticators $\mathcal{T}$, clients $\mathcal{C}$, and servers $\mathcal{S}$, i.e., $|\mathcal{P}| = |\mathcal{T}| + |\mathcal{C}| + |\mathcal{S}|$. Each party has a well-defined and non-ambiguous identifier, which one can think of as being represented as an integer. The state of authenticator 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) one or more registration contexts $\mathsf{st}_T.\mathsf{rct}_i$. A server S may also have multiple registration contexts $\mathsf{st}_S.\mathsf{rcs}_i$.

A PlA protocol consists of the following subprotocols:

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

Register This subprotocol is executed between an authenticator, a client, and a server. The server inputs a server name id_S, a username id_U, and a set of attestation public keys; the client inputs a server name $\hat{\mathsf{id}}_S$ and a username $\hat{\mathsf{id}}_U$; and the authenticator inputs its static storage. 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 an authenticator, a client, and a server. The server inputs the registration context for the intended username; the client inputs a server name $\bar{\mathsf{id}}_S$ and a username $\bar{\mathsf{id}}_U$; and the authenticator inputs its registration contexts. At the end of the protocol, the server accepts or rejects. The registration contexts of token and server may be updated after authentication (e.g., for sequence number updates).

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 registration or authentication protocol and return a challenge. The check algorithms get the same inputs, the challenge, and a response, then output accept or reject. The registration check additionally returns a registration context rcs, encoding user and server identities.

• 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 authenticator. These algorithms output commands denoted by M, which they generate from a challenge and the server and user identities.

• Authenticator-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. In the case of registration, this algorithm also outputs a registration context rct, encoding a server identity and optionally a user identity.

CORRECTNESS. Formally, correctness imposes that for any user names $\mathsf{id}_U, \hat{\mathsf{id}}_U, \bar{\mathsf{id}}_U$ and server name $\mathsf{id}_S, \hat{\mathsf{id}}_S, \bar{\mathsf{id}}_S$ the following probability is 1:

$$\Pr\left[\begin{array}{l|l} & (ak, vk) \xleftarrow{\$} \mathsf{Kg}(1^\lambda) \\ b = ((\mathsf{id}_S, \mathsf{id}_U) \stackrel{?}{=} (\hat{\mathsf{id}}_S, \hat{\mathsf{id}}_U) & c \xleftarrow{\$} \mathsf{rchallenge}(\mathsf{id}_S, \mathsf{id}_U, vk) \\ \quad \wedge (\mathsf{id}_S, \mathsf{id}_U) \stackrel{?}{=} (\bar{\mathsf{id}}_S, \bar{\mathsf{id}}_U)) & M \xleftarrow{\$} \mathsf{rcommand}(\hat{\mathsf{id}}_S, \hat{\mathsf{id}}_U, c) \\ \wedge\, \mathsf{rcs}.\mathsf{id}_S = \mathsf{id}_S & (r, \mathsf{rct}) \xleftarrow{\$} \mathsf{rresponse}(M, ak, G) \\ \wedge\, \mathsf{rct}.\mathsf{id}_S = \mathsf{id}_S & \mathsf{rcs} \leftarrow \mathsf{rcheck}(\mathsf{id}_S, \mathsf{id}_U, vk, c, r) \\ \wedge\, \mathsf{rcs}.\mathsf{id}_U = \mathsf{id}_U & c' \xleftarrow{\$} \mathsf{achallenge}(\mathsf{rcs}) \\ \wedge\, \mathsf{rct}.\mathsf{id}_U \in \{\bot, \mathsf{id}_U\} & M' \xleftarrow{\$} \mathsf{acommand}(\bar{\mathsf{id}}_S, \bar{\mathsf{id}}_U, c') \\ & r' \xleftarrow{\$} \mathsf{aresponse}(M', \mathsf{rct}, G') \\ & b \leftarrow \mathsf{acheck}(\mathsf{rcs}, c', r') \end{array}\right]$$

Token-side (rct) and server-side (rcs) registration contexts should store the identity of the server involved in the challenge response, as well as the user identity that the server used as input. We denote these as $\mathsf{rct}.\mathsf{id}_U$, $\mathsf{rct}.\mathsf{id}_S$, $\mathsf{rcs}.\mathsf{id}_U$, and $\mathsf{rcs}.\mathsf{id}_S$.

Intuitively, correctness imposes that the server always accepts an authentication that is consistent with a prior registration, and the registration on the token side produces the correct server name and optionally the correct user name. (Here we do not mean that the user can necessarily confirm that the correct names have been agreed, but rather that the internal state of the token encodes the correct names.)

7.2 Security Model

Trust model. Before defining security we clarify that there are no security assumptions on the communication channels shown in Figure 2. Again, the authenticator is assumed to be tamper-proof, so the model will not consider corruption of its 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. Finally, we require authenticators and servers to keep a unique registration context for each $(\mathsf{id}_U, \mathsf{id}_S)$ pair.

We now formally define security of PlA protocols.

Session oracles. Similar to PACA protocols, 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 authenticators, session oracles are structured as follows: $\pi_S^{i,0}$ refers to a pending or completed registration session, whereas $\pi_S^{i,j}$ for $j \geq 1$ refers to the j-th pending or

completed authentication session associated with $\pi_S^{i,0}$ after this registration completed. For tokens, the static storage (attestation material) is maintained by the security experiment. The security experiment also manages the attestation public keys of different authenticators and provides them to the server session oracles as needed.

Security experiment. The security experiment is run between a challenger and an adversary $\mathcal{A}$. In the beginning of the experiment, the challenger runs $(ak_T, vk_T) \xleftarrow{\$} \mathsf{Kg}(\,)$ 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:

- $\mathsf{Start}(\pi_S^{i,j}, \mathsf{id}_S, \mathsf{id}_U, T)$. The challenger instructs $\pi_S^{i,j}$ to execute 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 indicated server and user names and generate a challenge c, which is given to the adversary. We set $\pi_S^{i,j}.c$ as the returned challenge.
- $\mathsf{Challenge}(\pi_T^{i,j}, M)$. The challenger delivers command M to $\pi_T^{i,j}$, which proceeds to process the command using rresponse (if $j = 0$) or aresponse (if $j > 0$) and return the result R to the adversary. We define $\pi_T^{i,j}.M$ as the input M and $\pi_T^{i,j}.R$ as the output R. We define $\pi_T^{i,0}.\mathsf{rct}$ as the resulting registration context, which may be updated every time it is used for subsequent authentications.
- $\mathsf{Complete}(\pi_S^{i,j}, R)$. This query delivers an authenticator response to a server oracle, which proceeds 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. We define $\pi_S^{i,0}.\mathsf{rcs}$ as the resulting registration context, which may be updated every time it is used for subsequent authentications.

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 session $\pi_S^{i,0}$ and an authenticator registration session $\pi_T^{k,0}$ are partnered if they agree on a session identifier. A server authentication session $\pi_S^{i,j}$, for $j > 0$, and a token authentication session $\pi_T^{k,l}$ for $k > 0$ are partnered if: i) they agree on a session identifier; and ii) $\pi_S^{i,0}$ and $\pi_T^{k,0}$ are partnered.

The session identifier must be defined by the protocol. 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 cannot be used to authenticate under another registration context or server.

Advantage measure. We define the passwordless authentication advantage $\mathbf{Adv}_{\Pi}^{\mathsf{pla}}(\mathcal{A})$ as the probability that a server session accepts and it is not uniquely partnered with a token session: there must exist a unique token session which has derived the same session identifier, and no other server session has derived the same session identifier.

8 The W3C Web Authentication Protocol

In this section, we present the W3C's Web Authentication (WebAuthn) protocol [14] of FIDO2 following our PlA protocol syntax and analyze its seurity.[20]

Protocol Description. WebAuthn supports two types of operations: Registeration and Authentication (cf. Figure 1 and Figure 2 in [14]), respectively corresponding to the Register and Authenticate algorithms of a PlA protocol. The username space consists of human-palatable strings for user accounts. The server ID space consists of effective domains (e.g., hostnames) of server URLs. The key generation algorithm Kg is defined as part 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 [27].) Let H denote the SHA-256 hash function and h_c denote the challenge hash $H(ch)$. The core cryptographic operations are presented in Figure 6. WebAuthn is clearly a PlA protocol syntactically. It does not

[20] We do not include the WebAuthn explicit reference to user interaction/gestures at this point, as this is handled by the PACA protocol.

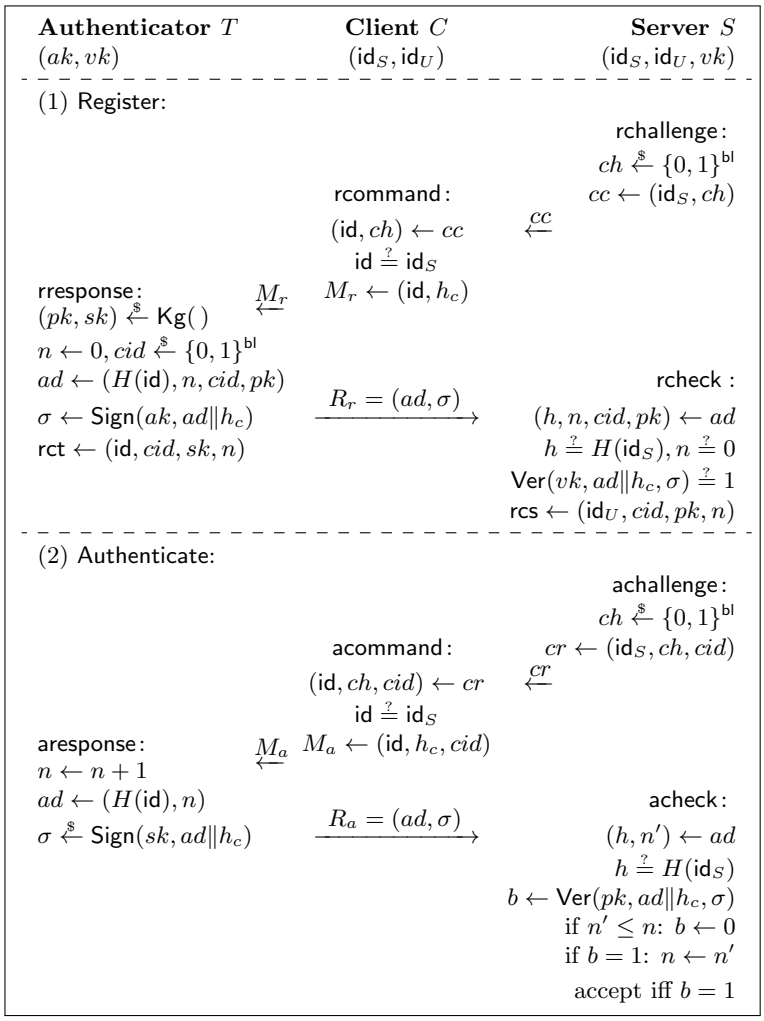

Figure 6: The WebAuthn protocol

give any information about user names to the token, which means that there is no binding between user name and registration except that which is carried out by the server. Adding this binding could be easily done by letting the token sign the username as well.

Security Results. The following theorem assesses PlA security of WebAuthn, which is proved in Appendix E using (ad, h_c) as the session identifier.

Theorem 3 *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:*

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

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 [12, 25] and the assumption that SHA-256 is collision-resistant.

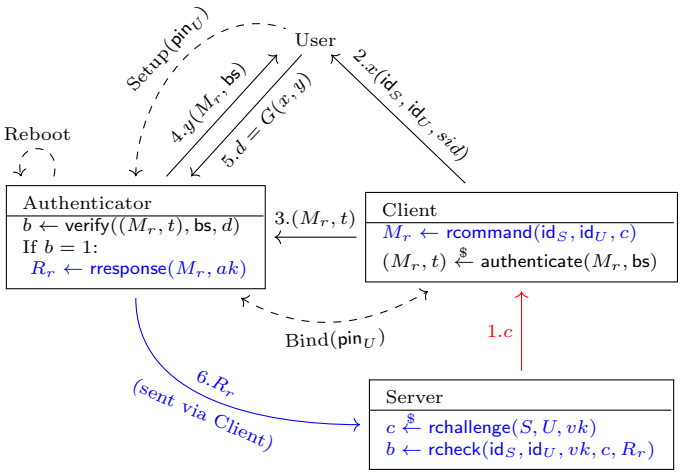

Figure 7: Full PACA+PlA registration flow: black = PACA, blue = PlA, red = authenticated (e.g., TLS), dashed = PACA algorithms/subprotocols.

9 The Composed Security of PACA and PlA

In this section we discuss the composed security of PACA and PlA and the implications of this composition for CTAP2*+WebAuthn. The composed protocol, which we refer simply as PACA+PlA 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 S when it gets a challenge from the server). It then authenticates 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 authentication 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 F 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 PACA+PlA, 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 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 F). We define a security property called User Authentication for the composed protocol. We analyze the PACA+PlA composition in a trust model is 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 attacker 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 allow the attacker to pick arbitrary usernames for both the client and server inputs, but 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 given to the attacker. Compromise of binding states and corruption of user pins is 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 and client sessions 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 and (optionally) the user name (the latter depends on whether the PlA protocol gives this guarantee).

We now give a short intuition on how we prove this result assuming the underlying PACA and PlA 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 user and server names $(\mathsf{id}_U, \mathsf{id}_S)$:

- 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.
- 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 F.

Security in the SUF model. The above result implies that the sPACA protocol from Section 6 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 CTAP2*+WebAuthn. The above result also implies that CTAP2*+WebAuthn 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 power-up, 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 CTAP2*+WebAuthn 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 user and server names used in a PlA session are the intended ones. 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.[21] In this case we get a strong guarantee that the token registered a credential or authenticated in the server with identifier $\mathsf{id}_S^\star$ under username $\mathsf{id}_U^\star$, where $(\mathsf{id}_U^\star, \mathsf{id}_S^\star)$ were 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 specific $(\mathsf{id}_U^\star, \mathsf{id}_S^\star)$ values 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 $(\mathsf{id}_U, \mathsf{id}_S)$ 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 power-up, as then there is no ambiguity as to which channel identifier is used and only a single client is providing user and server names 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: CTAP2 and WebAuthn, and the overall FIDO2 protocol 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 passwordless user authentication protocols.

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.

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

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

[10] M. Bellare, T. Kohno, and C. Namprempre, "Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the encode-then-encrypt-and-mac paradigm," *ACM Transactions on Information and System Security (TISSEC)*, vol. 7, no. 2, pp. 206–241, 2004.

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

[12] ——, "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.

[13] 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.

[14] 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.

[15] 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.

[16] 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

[17] 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.

[18] 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.

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

[20] 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.

[21] 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

[22] 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.

[23] 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

[24] 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.

[25] 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

[26] 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

[27] 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

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

[29] 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.

[30] 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.

[31] 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 Message Authentication Code

For a (deterministic) *message authentication code (MAC)* $\mathsf{MAC} : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^n$, 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 the MAC oracle $\mathsf{MAC}(k,\cdot)$. In the end, $\mathcal{A}$ outputs a message-tag pair (m,τ). Its advantage measure $\mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{MAC}}(\mathcal{A})$ is defined as the probability that $\mathsf{MAC}(m) = \tau$ and m was not queried to the $\mathsf{MAC}(k,\cdot)$ oracle.

MAC is *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.3 Collision-Resistant Hash Function Family

Consider a function family $H = \{h_k : \mathcal{D}_k \to \mathcal{R}_k\}_k$ generated by some algorithm $G(1^\lambda)$, where $\forall k \xleftarrow{\$} G(1^\lambda), |\mathcal{D}_k| > |\mathcal{R}_k|$ and computing h_k on any input is efficient given k. In the security experiment, an adversary $\mathcal{A}$ takes as input 1^λ and a random $k \xleftarrow{\$} G(1^\lambda)$, then outputs two messages $(x_1, x_2) \in \mathcal{D}_k$. 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 typically a single hash function instead of a function family.

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 $\mathsf{Sign}(sk,\cdot)$. 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 *existentially unforgeable under chosen message attack (EUF-CMA)* 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 a client/server or initiator/responder, but in CTAP2 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 shared 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 PAKE protocols (non-augmented) with perfect forward secrecy (PFS) and explicit mutual authentication, as described for example in [3, 4], 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].

PAKE PFS security experiment. An efficient adversary $\mathcal{A}$ is challenged to distinguish established session keys from truly random ones with an advantage that is better than password guessing. The challenger emulates an execution environment in which tokens $T \in \mathcal{T}$ and clients $C \in \mathcal{C}$ communicate securely 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 any global public parameters (CRS) that the protocol may rely on and samples passwords for all pairs of parties from a 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 characterises the most likely password.[22] The challenger manages a set of instances π_P^i, each corresponding to the state of session instance j at party $P \in \mathcal{C} \cup \mathcal{T}$, according to the protocol definition. The adversary is then executed with the CRS as input; it may 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 instance π_P^i (or creates this state if the instance was not yet initialized) and returns any outgoing messages to the attacker.
- CORRUPT: Given a pair of party identities (C, T), this oracle returns the corresponding pre-shared password.
- REVEAL: Given a party identity P and an instance i, this oracle checks π_P^j and, if this session instance has completed as defined by the protocol, the output of the session (usually either a secret key or an abort symbol) is returned to the attacker.
- TEST: Given a party identity P and an instance i, this oracle checks π_P^j and, if this session instance has completed as defined by the protocol *and* this session instance is *fresh*, the adversary is challenged on guessing bit b: if $b = 0$ then the derived key is given to the attacker; otherwise a new random key is returned.

Eventually the adversary terminates and outputs a guess bit b'. The definition of advantage excludes trivial attacks via the notion of session freshness used in the TEST oracle.

[22] We use this definition to emphasize that one does not need to assume that the distribution of pin's 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.

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 is fresh if: a) the instance completed; b) the instance was not queried to TEST or REVEAL before; c) at least one of the following four conditions holds: i) the adversary behaved passively when completing that session; ii) there exists more than one partner instance; iii) no partner instance exists and the associated password was not corrupted prior to completion; and iv) a unique fresh partner instance exists (implies not revealed).

A PAKE protocol offers PFS if, for any efficient attacker $\mathcal{A}$ interacting with the above experiment, we have that

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

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

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

- if a client C accepts a session with token T, then T has a pending unique session (which is the unique partner of the client session) where it has derived the same key and session identifier;
- if a token T accepts a session with client C, then C has already accepted a unique session (which is the unique partner of the token session) with the same key and session identifier.

Otherwise, client and server output an explicit abort symbol. Note that this implies that any active attack launched by the adversary leads to a session abort with overwhelming probability, unless the attacker was able to guess the password.

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

C Proof of Theorem 1

IND-1$PA for deterministic encryption. We first introduce the *indistinguishability under one-time chosen and then random plaintext attack (IND-1$PA)* security of a deterministic encryption scheme $\mathsf{DE} = (\mathcal{K}, \mathsf{E}, \mathsf{D})$. This notion is used in our proofs and may be of independent interest when only random messages are encrypted.

Definition 1 (IND-1$PA) *We define a security experiment associated with an adversary $\mathcal{A}$. The experiment samples a random key $k \xleftarrow{\$} \mathcal{K}$ and a random bit $b \xleftarrow{\$} \{0,1\}$. Then, $\mathcal{A}$ is granted access to the following encryption oracles:*

- $\mathcal{O}_{\mathsf{LR}}(m_0, m_1)$*: returns* $\mathsf{E}(k, m_b)$*, where the input m_0, m_1 are of the same bit length. This oracle is queried only once (if ever) and has to be the first query.*
- $\mathcal{O}_{\$}(l)$*: samples a random message $m \xleftarrow{\$} \{0,1\}^l$ and returns $(m, \mathsf{E}(k, m))$.*
- $\mathcal{O}_{\$\mathsf{LR}}(l)$*: samples independent random messages $m_0 \xleftarrow{\$} \{0,1\}^l$, $m_1 \xleftarrow{\$} \{0,1\}^l$ and returns $(m_0, m_1, \mathsf{E}(k, m_b))$.*

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

Comparison to other IND-CPA security notions. Note that our IND-1$PA security is implied by the deterministic version of IND-CPA – *indistinguishability under distinct chosen-plaintext attack (IND-DCPA)* security [10] (where $\mathcal{O}_{\mathsf{LR}}$ can be queried multiple times as long as messages in each world never repeat), because oracles $\mathcal{O}_{\$}$ and $\mathcal{O}_{\$\mathsf{LR}}$ can be simulated using the $\mathcal{O}_{\mathsf{LR}}$ oracle. However, IND-1$PA does not imply IND-DCPA. In particular, CBC_0 is IND-1$PA secure (see Lemma 1 below), but it

is obviously not IND-CPA secure for long messages (e.g., of length more than one block size). On the other hand, IND-1$PA clearly implies *one-time* IND-CPA for which no other queries are allowed after the single $\mathcal{O}_{\mathsf{LR}}$ query. Such one-time IND-CPA security was defined as security against passive attacks by Cramer and Shoup [15]. Note that the other direction is not true, e.g., the one-time pad is one-time IND-CPA secure but not IND-1$PA secure.

IND-1$PA security of CBC_0. Now, we prove that CBC_0 is IND-1$PA secure (and hence also one-time IND-CPA secure) by modeling its underlying AES-256 cipher E as a PRF.

Lemma 1 *For any efficient adversary $\mathcal{A}$, there exist an efficient adversary $\mathcal{B}$ such that:*

$$\mathbf{Adv}^{\mathsf{ind\text{-}1\$pa}}_{\mathsf{CBC}_0}(\mathcal{A}) \leq 2\mathbf{Adv}^{\mathsf{prf}}_{E}(\mathcal{B}) + 2\binom{\frac{\mu}{\mathsf{bl}}}{2}2^{-\mathsf{bl}},$$

where μ is the total bit length of ciphertexts in response to all encryption oracle queries.

Proof: First, we replace the PRF E in both the left and right worlds with a purely random function $f : \{0,1\}^\lambda \rightarrow \{0,1\}^\lambda$. Note that it is straightforward to simulate either world of the IND-1$PA experiment of CBC_0 with oracles E_k (for random k) or f. Therefore, there exists an efficient adversary $\mathcal{B}$ against the PRF security of E such that the game differences caused by replacing E with f are bounded by $2\mathbf{Adv}^{\mathsf{prf}}_{E}(\mathcal{B})$.

Let $\widetilde{\mathsf{CBC}}_0$ denote the CBC_0 encryption scheme of which the underlying PRF is replaced with a purely random function. We are left to show that the left and right worlds of the IND-1$PA experiment of $\widetilde{\mathsf{CBC}}_0$ are computationally indistinguishable. It is sufficient to bound the probability of input collisions for f because the two worlds are perfectly indistinguishable if no collisions occur. Let p denote such collision probability. Since there are μ/bl blocks in each world, by a union bound, we have $p \leq \mu/\mathsf{bl}(\mu/\mathsf{bl}-1)2^{-\mathsf{bl}}$. We also refer to [9] (Lemma 18) for a more detailed security analysis of the randomized CBC scheme that also applies to our case. □

Proof of Theorem 1. We complete the final proof as follows.

Proof: We recall that, because the adversary is not allowed to make Connect queries, it is unable to create client-side session oracles that can be actively attacked.

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

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

Game 1: The challenger proceeds as before except it replaces all shared keys $K = \mathcal{H}(abG.x)$ established in Setup queries and Execute queries with independent random values $\widetilde{K} \xleftarrow{\$} \{0,1\}^\lambda$. By a hybrid argument, there exists an efficient adversary $\mathcal{B}$ against the SCDH security of $\mathbb{G}$ 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 a hybrid game, we embed the challenge group elements $\tilde{a}G, \tilde{b}G$ into the corresponding authenticator and client oracles and answer random oracle $\mathcal{H}$ queries via lazy sampling. Note that when receiving an arbitrary bG from $\mathcal{A}$ to the authenticator oracle via a Send query, we check if $\tilde{a}bG.x$ has already been queried to the random oracle $\mathcal{H}$ using the verification oracle $\mathcal{O}_{\tilde{a}}(\cdot,\cdot)$. This is sufficient to answer $\mathcal{H}$ queries consistently and set the resulting shared keys accordingly. Before setting $K = \mathcal{H}(\tilde{a}\tilde{b}G.x)$ to $\widetilde{K}$, we also need to check if $\tilde{a}\tilde{b}G$ has already been queried: if so we can win the SCDH game, otherwise we simulate $\mathcal{A}$'s view perfectly.

Game 2: The challenger proceeds as before except it replaces all transmitted ciphertexts $c_p = \mathsf{E}(\widetilde{K}, 0, \mathsf{pin}_U)$, $c_{ph} = \mathsf{E}(\widetilde{K}, 0, \mathcal{H}(\mathsf{pin}_U))$, $c_{pt} = \mathsf{E}(\widetilde{K}, 0, pt)$ in Setup and Execute queries with $\tilde{c}_p \leftarrow \mathsf{E}(\widetilde{K}, 0, \widetilde{\mathsf{pin}_U})$, $\tilde{c}_{ph} \leftarrow \mathsf{E}(\widetilde{K}, 0, \mathcal{H}(\widetilde{\mathsf{pin}_U}))$, $\tilde{c}_{pt} \leftarrow \mathsf{E}(\widetilde{K}, 0, \widetilde{pt})$, where $\widetilde{\mathsf{pin}_U} \xleftarrow{\$} \mathcal{PIN}, \widetilde{pt} \xleftarrow{\$} \{0,1\}^\lambda$ are independent random values. By a hybrid argument, there exists an efficient adversary $\widetilde{\mathcal{C}}$ against the IND-1$PA security of CBC_0 such that $|\Pr_1 - \Pr_2| \leq (q_{\mathsf{S}} + q_{\mathsf{E}}) \cdot \mathbf{Adv}^{\mathsf{ind\text{-}1\$pa}}_{\mathsf{CBC}_0}(\widetilde{\mathcal{C}})$.

To simulate a pair of consecutive hybrid games G_k, G_{k+1} (e.g., replacing the transmitted ciphertexts in Setup/Execute), we sample an independent random PIN $\widetilde{\mathsf{pin}_U} \xleftarrow{\$} \mathcal{PIN}$ and query the IND-1$PA encryption oracles of CBC_0 as follows. First, query $c^* \leftarrow \mathcal{O}_{\mathsf{LR}}(\mathsf{pin}_U, \widetilde{\mathsf{pin}_U})$ if the considered Setup/Execute query involves the authenticator setup phase. Then, if $\mathsf{pin}_U \neq \widetilde{\mathsf{pin}_U}$, query $(m_0, m_1, c_b) \xleftarrow{\$} \mathcal{O}_{\$\mathsf{LR}}(\,)$; otherwise (if $\mathsf{pin}_U = \widetilde{\mathsf{pin}_U}$), query $(m, c) \xleftarrow{\$} \mathcal{O}_{\$}(\,)$ and let $m_0 = m_1 = m$ and $c_b = c$. Finally, query $(m'_0, m'_1, c'_b) \xleftarrow{\$} \mathcal{O}_{\$\mathsf{LR}}(\,)$. Now, we set $c_p \leftarrow c^*$, $c_{ph} \leftarrow c_b$, $c_{pt} \leftarrow c'_b$, $pt \leftarrow m'_0$. It is not hard to see that, in the left world, the above ciphertexts and pt are equal to the original ones in game G_k while, in the right world, they are equal to the replaced ciphertexts that encrypt random values in game G_{k+1}. Note that we simulate the random oracle $\mathcal{H}$ via lazy sampling as before except that we fix $\mathcal{H}(\mathsf{pin}_U) \leftarrow m_0$ and $\mathcal{H}(\widetilde{\mathsf{pin}_U}) \leftarrow m_1$ (or $\mathcal{H}(\mathsf{pin}_U) \leftarrow m$ if $\mathsf{pin}_U = \widetilde{\mathsf{pin}_U}$) in the beginning.

Now, recall that encrypting a user PIN in the setup session yields a 4-block ciphertext and encrypting its hash in the binding session yields a 1-block ciphertext. According to Lemma 1, we have $|\Pr_1 - \Pr_2| \leq 2(q_{\mathsf{S}} + q_{\mathsf{E}}) \cdot \mathbf{Adv}^{\mathsf{prf}}_{E}(\mathcal{C}) + 2q_{\mathsf{S}}\binom{4}{2}/2^{\mathsf{bl}}$.

Game 3: The challenger proceeds as before, except it aborts if some authenticator oracle received a malicious c_{ph} from $\mathcal{A}$ (via a Send query to a token-side oracle) and the decrypted message matches its transformed PIN. Note that after Game 2, the authenticators' transformed PINs and the clients' input PINs are independent from $\mathcal{A}$'s view. Since each Setup query allows $\mathcal{A}$ to create a token on which it is able to make at most n_{max} guesses (or n_{th} guesses without requiring authenticator reboots) about the corresponding authenticator's transformed PIN and each guess succeeds with probability at most $1/2^{h_{\mathscr{D}}} + 1/2^{\mathsf{bl}}$ (where the second term accounts for a possible collision on the output of $\mathcal{H}$), by a union bound we have $|\Pr_2 - \Pr_3| \leq nq_{\mathsf{S}}(1/2^{h_{\mathscr{D}}} + 1/2^{\mathsf{bl}}) = nq_{\mathsf{S}}/2^{h_{\mathscr{D}}} + nq_{\mathsf{S}}/2^{\mathsf{bl}}$.

Game 4: The challenger proceeds as before except that it aborts if ever a token session accepts a command that was not issued by one of the token partners. This event can be reduced to the EUF-CMA security of the MAC scheme MAC by an efficient adversary $\mathcal{D}$. Adversary $\mathcal{D}$ first guesses the accepting authenticator T and its reboot cycle with probability at least $1/(q_{\mathsf{S}}q_{\mathsf{R}})$ (because each authenticator reboot generates an independent random pt), then simulates the game with the MAC oracle of MAC. This simulation is perfect except when collisions occur in the random oracle queries (used to get the message digest that is actually authenticated with the MAC), which happens with probability at most $(q_{\mathsf{A}} + q_{\mathcal{H}})^2/2^{\mathsf{bl}}$. Therefore, $|\Pr_3 - \Pr_4| \leq q_{\mathsf{S}}q_{\mathsf{R}} \cdot \mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{MAC}}(\mathcal{D}) + (q_{\mathsf{A}} + q_{\mathcal{H}})^2/2^{\mathsf{bl}}$.

Final analysis. In the final game it remains to establish that partnership relations are unique and refer to valid partner session oracles at the time a command is accepted/issued. Note that no binding is complete in this game except via Execute queries, and hence accepted commands come from client session oracles. Uniqueness therefore follows from a birthday bound on the order of the elliptic-curve group of the form ${q_{\mathsf{E}}}^2/q$. The validity condition is now trivial, as we know that token sessions only accept commands if they are valid, i.e. powered up, and we know it has a unique partner that was not involved in setup, and hence never goes invalid. □

D Proof of Theorem 2

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 real experiment for $\mathcal{A}$, so $\Pr_0 = \mathbf{Adv}^{\mathsf{suf}}_{\mathrm{sPACA}}(\mathcal{A})$.

Game 1: The challenger proceeds as before except it 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 the trusted authenticator setup phase, our proof no longer requires the SCDH assumption. Similar to the corresponding proof in Appendix C, there exists an efficient adversary $\mathcal{B}$ against the CDH security of $\mathbb{G}$ such that $|\Pr_0 - \Pr_1| \leq q_{\mathsf{S}}\mathbf{Adv}^{\mathsf{cdh}}_{\mathbb{G},g}(\mathcal{B})$.

Game 2: The challenger proceeds as before except it replaces all transmitted ciphertexts $c_p = \mathsf{E}(\widetilde{K}, 0, \mathsf{pin}_U)$ in Setup queries with $\tilde{c}_p \leftarrow \mathsf{E}(\widetilde{K}, 0, \widetilde{\mathsf{pin}_U})$ for some fixed pin $\widetilde{\mathsf{pin}_U}$. By a hybrid argu-

ment, there exists an efficient adversary $\widetilde{\mathcal{C}}$ against the one-time IND-CPA security of CBC_0 such that $|\Pr_1 - \Pr_2| \leq q_{\mathsf{S}} \mathbf{Adv}^{\mathsf{ot\text{-}ind\text{-}cpa}}_{\mathsf{CBC}_0}(\widetilde{\mathcal{C}})$.

To simulate a pair of consecutive hybrid games G_k, G_{k+1} (e.g., replacing c_p with $\tilde{c}_p$ in $\mathsf{Setup}(\pi^i_T, \pi^j_C)$), we replace the fixed PIN $\widetilde{\mathsf{pin}_U}$, query $c_b \leftarrow \mathcal{O}_{\mathsf{LR}}(\mathsf{pin}_U, \widetilde{\mathsf{pin}_U})$, and set $c_p \leftarrow c_b$. Recall that IND-1$PA security implies one-time IND-CPA security. Similar to the proof in Appendix C, from Lemma 1 we have $|\Pr_1 - \Pr_2| \leq 2q_{\mathsf{S}} \cdot \mathbf{Adv}^{\mathsf{prf}}_E(\mathcal{C}) + 12q_{\mathsf{S}}/2^{\mathsf{bl}}$.

A PAKE attacker. We now describe a PAKE attacker $\mathcal{D}(\mathcal{A})$ that we will use to bridge the following two hops. The adversary is parameterized with a bit c that instructs it to attack the explicit authentication property (which we use hop to game 3) or the key secrecy property (which we use to hop to game 4). Here we model the password derivation hash as a random oracle. $\mathcal{D}$ uses the PFS SEND oracle to simulate the PACA Execute, Connect and Send sessions, mapping compromise queries to REVEAL queries and pin corruption queries password corruption queries (given a password $\mathcal{D}$ can use its simulation of the random oracle to keep consistency with PINs up to a statistical birthday bound that depends on the number of random oracle queries $q_{\mathcal{H}}$ to account for hash collisions). For the sessions whose PACA binding states are fresh when they are used in the PACA simulated experiment, $\mathcal{D}$ proceeds differently depending on bit c: if $c = \top$, $\mathcal{D}$ uses the TEST oracle in the PFS game; otherwise it uses the REVEAL oracle. $\mathcal{D}$ aborts if ever explicit authentication is broken in the PAKE game (as this means it already won). When $\mathcal{A}$ terminates, if $c = \top$, $\mathcal{D}$ returns 1 if $\mathcal{A}$ won the PACA game and 0 otherwise. Otherwise it just returns a random bit.

We now analyze the maximum number of SEND queries placed by this adversary. For this, we assume it does not abort. To compute this bound we assume that the PAKE protocol 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 our suggested instantiations.

Since each token allows $\mathcal{A}$ to make at most n_{max} failed binding attempts (or n_{th} guesses without requiring authenticator reboots) about the password (i.e., $\mathcal{H}(\mathsf{pin}_U)$) by interacting with an authenticator, we know that $\mathcal{D}$ will make at most $2nq_{\mathsf{S}}$ 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 $\mathcal{D}$ makes at most $2nq_{\mathsf{C}}$ queries to deal with client-side Send queries by $\mathcal{A}$. In total $\mathcal{D}$ makes at most $2(nq_{\mathsf{S}} + q_{\mathsf{C}})$ SEND queries where it actively attacks the session,

Game 3: The challenger proceeds as before, except it aborts any time the explicit authentication guarantee is violated on either the client side or the token side. We bound the adversary's change of advantage in this hop by running attacker $\mathcal{D}$ above in its authentication adversary ($c = \perp$) mode. It is clear that the two games are identical until bad and that $\mathcal{D}$ wins the PAKE explicit authentication game if ever this event occurs, which means we have: $|\Pr_2 - \Pr_3| \leq 2(nq_{\mathsf{S}} + q_{\mathsf{C}})/2^{h_{\mathscr{D}}} + \epsilon_{\mathsf{ea}} + {q_{\mathcal{H}}}^2/2^{\mathsf{bl}}$.

Game 4: The challenger proceeds as before except it replaces PAKE secret keys associated with PACA binding sessions for which the password was not corrupted prior to completion with completely random keys. We bound the adversary's change of advantage in this hop by running attacker $\mathcal{D}$ above in its key secrecy adversary ($c = \top$) mode. Adversary $\mathcal{D}$ perfectly interpolates between the two games, which means we have have that $|\Pr_3 - \Pr_4| \leq 2(nq_{\mathsf{S}} + q_{\mathsf{C}})/2^{h_{\mathscr{D}}} + \epsilon_{\mathsf{pfs}} + {q_{\mathcal{H}}}^2/2^{\mathsf{bl}}$.

Game 5: The challenger proceeds as before except that it aborts if ever a token session accepts a command that was not issued by its unique partner. This bad event can be reduced to the EUF-CMA security of the MAC scheme MAC by an efficient adversary $\mathcal{D}$. $\mathcal{E}$ first guesses the accepting authenticator oracle π^i_T with probability at least $1/(q_{\mathsf{E}} + q_{\mathsf{C}})$, then simulates the game with the MAC oracle of MAC. Recall that PAKE guarantees client authentication, i.e., π^i_T receives a random key (in an uncompromised session) only if there is a partner client oracle that received the same key. This ensures that $\mathcal{D}$ can simulate the MAC operations of π^i_T and its partner perfectly. Therefore, we have $|\Pr_4 - \Pr_5| \leq (q_{\mathsf{C}} + q_{\mathsf{E}}) \cdot \mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{MAC}}(\mathcal{E})$.

Final analysis. In the final game we know partnerships are unique. The validity condition is now trivial, as we know that token sessions only accept commands if they are valid and PAKE sessions are

all valid on the client side. □

Token-side (rct) and server-side (rcs) registration contexts should store the identity of the server involved in the challenge response, as well as the user identity that the server used as input. We denote these as $\mathsf{rct.id}_U$, $\mathsf{rct.id}_S$, $\mathsf{rcs.id}_U$, and $\mathsf{rcs.id}_S$.

Intuitively, correctness imposes that the server always accepts an authentication that is consistent with a prior registration, and the registration on the token side produces the correct server name and optionally the correct user name. (Here we do not mean that the user can necessarily confirm that the correct names have been agreed, but rather that the internal state of the token encodes the correct names.)

E 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 real experiment for $\mathcal{A}$, so $\Pr_0 = \mathbf{Adv}^{\mathsf{pla}}_{\mathrm{WebAuthn}}(\mathcal{A})$.

Game 1: The challenger proceeds as before, except it aborts if a hash collision occurs. By definition, there exists an efficient adversary $\mathcal{B}$ against the collision-resistance security of H such that $|\Pr_0 - \Pr_1| \leq \mathbf{Adv}^{\mathsf{coll}}_H(\mathcal{B})$.

Game 2: The challenger proceeds as before except it aborts if there exists a server that generates two identical challenges in authentication sessions. We have $|\Pr_1 - \Pr_2| \leq +{q_\mathsf{S}}^2/2^{\mathsf{bl}}$. Note that at this point we excluded any possibility of two session identifiers colliding on the server side.

Game 3: The challenger proceeds as before except it aborts if a server-side oracle does not have a unique token-side partner. The associated bad event can be reduced by by 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 and public key of Sig. There are two cases, $\pi_S^{i,j}$ has 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 ${q_\mathsf{C}}^2/2^{\mathsf{bl}}$ 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 accepted because it successfully verified a signature on a public-key pk that was established in registration session $j = 0$. Because the signature counter n is incremented for every new session, $\pi_S^{i,j}$ cannot have two partners that signed the same message, except if the same credential pk used by $\pi_S^{i,j}$ was generated in two registration sessions. If this happens, $\mathcal{C}$ can trivially forge a signature to a new message as it knows the signing key. Otherwise, since the response issued by tokens includes the counter n, it is impossible to have two partners. On the other hand, if $\pi_S^{i,j}$ has no partner (note 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 $|\Pr_2 - \Pr_3| \leq q_\mathsf{C} \cdot \mathbf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{Sig}}(\mathcal{C}) + {q_\mathsf{C}}^2/2^{\mathsf{bl}}$.

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. □

F Formal Composition Result for PACA+PlA

In this section we discuss the combined security of PACA and PlA as a black box, and then analyze the implications for the CTAP2*+WebAuthn and sPACA+WebAuthn instantiations. The composed protocol, which we refer simply as PACA+PlA 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.

F.1 Protocol Syntax

The state of authenticator T, denoted st_T is partitioned into the following components: i) static PACA storage $\mathsf{st}_T.\mathsf{ss}$; ii) static PlA attestation key pair $(\mathsf{st}_T.ak, \mathsf{st}_T.vk)$; iiI) static PlA registration contexts, which we denote by $\mathsf{st}_T.\mathsf{rct}_i$ (together denoted by $\mathsf{st}_T.\mathsf{rct}$); vi) volatile power-up or reset state $\mathsf{st}_T.\mathsf{rs}$; v) one or more volatile binding states $\mathsf{st}_T.\mathsf{bs}_i$. A client C may have multiple binding states, which we denote by $\mathsf{bs}_{C,j}$. A server S may have multiple registration contexts, which we denote by $\mathsf{rcs}_{S,k}$.

A PACA+PlA 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.

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.

Reboot This algorithm represents a power-down/power-up 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.

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 power-up 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 power-up state (e.g., a counter).[23] If the protocol fails, no binding states are set, but the authenticator power-up state may still be updated. We assume the client always initiates this protocol once it gets the PIN from the user.

Register This subprotocol is executed between the four parties. The server inputs a server name id_S, a username id_U, and a set of attestation public keys; the client inputs a server name $\hat{\mathsf{id}}_S$, a username $\hat{\mathsf{id}}_U$, 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 the registration context $\mathsf{rcs}_{S,k}$ for the intended username; the client inputs a server name $\bar{\mathsf{id}}_S$, a username $\bar{\mathsf{id}}_U$, 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 outputs accept or reject.

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.

[23] 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.

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 CTAP2*+WebAuthn and it allows us to have a more intuitive security definition. It is clear from the syntax above that PACA+PlA inherits without change the Key Generation algorithm from PlA and the Setup, Reboot, and Bind subprotocols from PACA. The interaction between the PACA and PlA algorithms is visible only in the Register and Authenticate subprotocols, which like for PlA we restrict to the following two-pass protocol that uses PACA and PlA:

- Server-side computation is split into four procedures: rchallenge and rcheck for registration, 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 rcommand or acommand, encoding the results as M_r and M_a, respectively. Finally it uses PACA to obtain an authenticated command that can be sent to the token.

- Authenticator-side computation does the obvious thing: on input an authenticated 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 rresponse or 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.

F.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 performs the combined parameter generation of both the PACA and PlA 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 Authenticate and Verify, which are replaced by the following queries:

• $\mathsf{Start}(\pi_S^{i,j}, \pi_C^k, \mathsf{id}_U, \mathsf{id}'_U, T)$. 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 the indicated user name id_U 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 $\mathsf{rcommand}(\mathsf{id}_S, \mathsf{id}'_U, c)$ (if $j = 0$) or $\mathsf{acommand}(\mathsf{id}_S, \mathsf{id}'_U, c)$ (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.
• Complete($\pi_S^{i,j}, R$). This query delivers an authenticator response to a server oracle, which proceeds 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 PACA+PlA 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, authenticated commands (M_r, t_r) and (M_a, t_a), username U, 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.
8. The registration context of $\pi_S^{i,0}$ encodes some username id_U, which was the server-side username input; If the registration context of $\pi_T^{k,0}$ encodes username $\mathsf{id}'_U \neq \perp$, then $\mathsf{id}'_U = \mathsf{id}_U$ and id_U was the client-side input to π_C^x and $\pi_{C'}^y$.

Note that here id_U is fixed by the server and 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)[24] 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 4 *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.*

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

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_C^x and $\pi_{C'}^y$ must in fact have received the single challenges produced by $\pi_S^{i,0}$ and $\pi_S^{i,j}$, 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 CTAP2*+WebAuthn 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).