Privacy and Security of FIDO2 Revisited

Manuel Barbosa
Universidade do Porto (FCUP) &
INESC TEC & Max Planck Institute
for Security and Privacy
Portugal, Germany
mbb@fc.up.pt

Alexandra Boldyreva
Georgia Institute of Technology
USA
sasha@gatech.edu

Shan Chen*
Southern University of Science and
Technology
China
chens3@sustech.edu.cn

Kaishuo Cheng
Georgia Institute of Technology
USA
kcheng89@gatech.edu

Luís Esquível
Universidade do Porto (FCUP) &
INESC TEC
Portugal
luis.esquivel.costa@gmail.com

Abstract

We revisit the privacy and security analyses of FIDO2, a widely deployed standard for passwordless authentication on the Web. We discuss previous works and conclude that each of them has at least one of the following limitations: (i) impractical trusted setup assumptions, (ii) security models that are inadequate in light of state of the art of practical attacks, (iii) not analyzing FIDO2 as a whole, especially for its privacy guarantees. Our work addresses these gaps and proposes revised security models for privacy and authentication. Equipped with our new models, we analyze FIDO2 modularly and focus on its component protocols, WebAuthn and CTAP2, clarifying their exact security guarantees. In particular, our results, for the first time, establish privacy guarantees for FIDO2 as a whole. Furthermore, we suggest minor modifications that can help FIDO2 provably meet stronger privacy and authentication definitions and withstand known and novel attacks.

Keywords

FIDO2, CTAP2, WebAuthn, Privacy, Authentication

1 Introduction

The fast-growing adoption of FIDO2 [14], actively supported by software giants Microsoft, Google and Apple, makes it a de-facto standard for passwordless authentication. FIDO2 is maintained by the FIDO (Fast IDentity Online) Alliance[1], a community of stakeholders that manages the specifications of the protocol and promotes its adoption.

The basic flows of FIDO2 are shown in Figure 1, inspired by the corresponding figure in [3]. At its core, FIDO2 is composed of two sub-protocols: WebAuthn (W3C's Web Authentication) [24, 25] and CTAP2 (Client to Authenticator Protocol version 2.x) [1, 13].

*Affiliated with both the Research Institute of Trustworthy Autonomous Systems and the Department of Computer Science and Engineering
[1]https://fidoalliance.org

Proceedings on Privacy Enhancing Technologies 2025(3), 304–332
© 2025 Copyright held by the owner/author(s).
https://doi.org/10.56553/popets-2025-0100

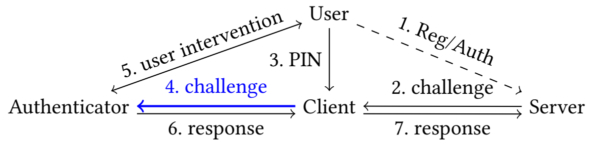


Figure 1: The simplified FIDO2 flow adapted from Fig. 2 in [3], where the CTAP2 authorized command is highlighted in blue. The dashed line means the communication with the server is established through the client.

WebAuthn specifies how a user can *register* a credential (a public signature verification key) at a server—associating it to a new or an existing account—and later rely solely on the corresponding private signing key for passwordless *authentication*. A *user*, in this context, normally refers to a human, who uses a *client* (typically a browser) to interact with the *server*. Cryptographically, an authentication run consists of a challenge-response exchange, in which the server issues a challenge and then checks if this challenge has been correctly digitally signed (along with relevant public metadata) by a public key that identifies the user. The private signing keys associated with WebAuthn credentials are often stored in secure hardware devices called *authenticators* (or *tokens*). Registration runs allow a user to create a server-specific public-key and associate it with an account at the server. New credentials uploaded to the server may also be signed using a (long-term) attestation private key that guarantees that the credential has indeed been generated by a secure device. This process is referred to as *attestation* and WebAuthn supports several different attestation modes.

CTAP2 (simply referred to as CTAP in this paper) specifies an access-control protocol that allows a client to issue authorized commands to unlock a FIDO2 authenticator with a set of permissions, which may allow creating a new credential for registration or using an existing credential for authentication. Access is granted only after human intervention, e.g., providing a PIN to the client, pressing a button on the token (known as *user presence*), providing biometric information to the token (known as *user verification*), etc.

Known security results and open problems. The design rationale and threat model for FIDO2 are described informally by the FIDO Alliance in [14]. The privacy properties of FIDO2 have been formally analyzed in [7, 16, 17] but they, so far, focus only on WebAuthn. A formal view of FIDO2's authentication properties has been established by a sequence of research papers that focused, first on the provable security of WebAuthn [16] and, more recently, on the overall FIDO2 as composed by WebAuthn and CTAP [3, 6]. While these works offered comprehensive studies of FIDO2's provable security, there are still gaps in the results they provided, both for privacy and authentication goals.

Results on FIDO2 privacy. User privacy is one of the key features claimed by FIDO, but it has received less attention from researchers compared to authentication. Hanzlik *et al.* [17] formally defined privacy (unlinkability) to analyze the WebAuthn component. Their definition assures that different registrations do not reveal if they are performed using the same token, and hence different interactions of the tokens are unlinkable. They proved that WebAuthn provides user privacy. Their work focused on non-resident keys—a variant of the protocol where the token keeps only a symmetric key and re-generates a signing key whenever it is needed for authentication—and does not consider attestation. Bindel *et al.* [7] extended their privacy results to cover several attestation modes and focused on resident keys, i.e., the case where the token stores authentication signing keys securely. However, their work also considered privacy guarantees of WebAuthn only.

We stress that at this point, to the best of our knowledge, there is no prior analysis of FIDO2 that considered the potential impact of the CTAP component on user privacy[2]. It was acknowledged in [17] that metadata can be used to link interactions of the token, but the authors do not consider data exchanged outside of WebAuthn. This means that, for example, it is not immediately clear whether CTAP preserves user privacy, given that a user may reuse the same PIN across different tokens. Moreover, the communication with the token may reuse meta-information or cryptographic material, potentially breaking unlinkability.

Results on FIDO2 authentication. On the authentication front, a recent work [4] demonstrated several man-in-the-middle attacks on FIDO2 USB tokens that are launched by wrapping the system library that the client browser uses to exchange messages with the hardware token. The attacks are deployed using a simple malware that does *not* require privileged access to the user's machine. In one type of attacks called *rogue key attacks*, only the final token-to-client message in a registration run is replaced, causing an *uncompromised* client (also called an *honest* client) to send to the server a public key that is actually under the attacker's control. Such attacks are possible because FIDO2 credentials sent from a token to a client are not cryptographically protected. Even more recently, the authors in [9] demonstrated successful message injection attacks on a USB bus by *another USB device*. Interestingly, the reported attacks coincide exactly with what is needed to launch a rogue key attack on FIDO2: first, monitor USB communication until the very end of the protocol and, second, override the final message sent by the target USB authenticator back to the host machine.

How can the existing provable security results for FIDO2 co-exist with rogue key attacks? The work [3] assumed that each token has a unique attestation private key, whose paired attestation public key is known to the servers a priori, so that rogue key attacks do not apply. However, this assumption does not reflect the practical setup: either no attestation is used, or many tokens share the same long-term attestation key pair (which is an imperative to prevent tracking and guarantee some form of user privacy). The work [6] considered only attackers that are passive during registration, which is not realistic as the aforementioned attacks demonstrate. The work [7] considered only the WebAuthn protocol, which means that the security of client-token communications is not considered. Finally, the full version of [3] considered only *batch attestation* (also known as Basic attestation mode), where many tokens share the same attestation key pair. However, in this case, a server cannot distinguish tokens from the same batch, which leaves open the possibility for an attacker to launch a rogue key attack with a valid token from the same batch as the user's token.

Our contributions. Our work closes the gaps we identified in prior work. For both security goals of privacy and authentication, our analysis is *modular*: we provide the security models and proofs for CTAP and WebAuthn separately, and then show how they compose to imply the security of the whole FIDO2. Our detailed contributions are listed as follows.

Privacy. We present the following new results on FIDO2 privacy:

- We propose a new security model for user privacy (anonymity). The goal of privacy is to ensure that multiple registrations involving the same token cannot be linked together, in order to avoid tracking. Unlike prior works [7, 17], we consider a more powerful attacker, which can compromise the servers and moreover, collaborate with local network attackers working at the CTAP level. Our security model allows to capture scenarios where an attacker can access and manipulate CTAP communications (e.g., when the token is lost, stolen or inserted into a corrupt USB hub or an untrusted machine) and wants to determine if it has observed some protocol execution that involved this authenticator or even has interacted with it in the past. Moreover, this attacker could be in collusion with a malicious server.
- We analyze FIDO2 in our privacy model. We observe that, formally, its privacy could be compromised by the reuse of ephemeral cryptographic parameters (Diffie-Hellman shares). This can happen when users register or authenticate multiple accounts without rebooting the token (by remaining plugged-in and not putting the computer to sleep). Our privacy definitions also highlight potential risks to user privacy that may be enabled by meta-information that is exposed by the token to the client, in excess of what CTAP specifies. For example, leakage of the contents in error messages or device identification information may lead to trivial breaks of the privacy guarantees of FIDO2 in our model.
- We prove that, with minor protocol changes that prevent the above Diffie-Hellman (DH) share reuse and by enforcing no leakage of the above meta-information, the cryptographic design of FIDO2 indeed guarantees strong privacy properties for the user, even considering local network attackers. We also show that the current FIDO2 guarantees privacy, as long as the attacker does not observe CTAP traces where a token reuses the same DH share when interacting with clients to register multiple accounts. The

[2]Kepkowski *et al.* [19] identified a timing attack that links user accounts across services, exploiting implementation flaws on hardware tokens. This is an attack on a full implementation of FIDO2. However, since it is based on side-channel analysis, is does not directly relate to the cryptographic notions of unlinkability we consider here.

take-away message from these results is that the cryptographic traces produced by CTAP do not undermine privacy as long as DH shares are not repeated between usages. We do not argue that the concrete attack scenarios where DH share reuse could be exploited to break privacy are a significant reason for concern. Our claim here is that our results establish the first leakage upper bound for CTAP (and FIDO2 as a whole): if CTAP privacy attacks contribute to compromise FIDO2 privacy, then it can only be due to DH share reuse. Additionally, our results show that CTAP authentication security also plays a crucial role in the overall privacy guarantees of FIDO2. Intuitively, CTAP access control prevents tokens from being *unlocked* and then tracked, e.g., an attacker could unlock a stolen token and check if any of its stored signing keys is associated with a given credential observed elsewhere.

Authentication. We present the following new results on FIDO2 authentication:

- We present a counterexample to the security claim for CTAP in [6] in the form of an attack that breaks the protocol in the security model for authentication adopted by [6]. The attack highlights a subtle oversight in their security proof and effectively shows that their model is too strong, and that FIDO2 can only be proved secure in a slightly weaker model.
- We fix the security definitions under which the current version of FIDO2 can be proved to provide passwordless authentication. Our model is similar to that of [3, 6] and guarantees the following: (i) a server *registers* only credentials generated by valid *physical* authenticators, (ii) if the server successfully *authenticates* a registered credential, then the token that generated that credential must be involved in the authentication, and (iii) if a token is interacting with an honest client, then it will only answer registration/authentication requests if the client unlocks this token using the correct user PIN.[3] We give a detailed proof of FIDO2 authentication security in this model that addresses the shortcomings we identified in prior work. Our proof applies to the attestation modes most commonly used by USB tokens: None, Self and Basic.
- We propose a stronger variant of the authentication model that captures rogue key attacks and USB injection attacks demonstrated in [4, 9]. Intuitively, this stronger model further guarantees that (iv) if a server successfully *registers* or *authenticates* a credential via an honest client, then the user must have *authorized* that specific client to unlock the token that generated that credential. Crucially, this model does *not* assume TOFU: the client (browser) plays a role in ensuring the server (and to the user) that, if the server is talking to a client through an authenticated channel (e.g., a TLS connection), then only a token unlocked by this specific client can successfully register a credential.
- We propose a simple fix to CTAP that appends a message authentication code (MAC) to the token responses before they are sent to the client (and then delivered to the server). We then prove that, with this fix, FIDO2 is provably resistant to rogue key attacks and meets our stronger security. Again, our results apply to the aforementioned attestation modes: None, Self and Basic.

[3]Note that, as in prior work and due to the lack of token-to-client authentication, FIDO2 guarantees only that the *same* token is used in registration and authentication. However, it does not ensure that the token used in authentication is bound to the (user-controlled) client through which the authentication run is carried out, even when assuming *trust on first use (TOFU)*, i.e., the attacker was passive during registration.

Responsible disclosure. We communicated our findings to the FIDO Alliance.

Structure of this paper. In the next section we expand on the background and relation to prior work. Then, we describe the formal protocol syntax that captures CTAP, as most of our contributions focus on CTAP analyses. In Section 4 we start with the analysis of authentication properties of FIDO2, since our privacy results depend on it. In Section 5 we focus on the privacy properties of the protocol. In Section 6 we discuss the practical implications of our proposed fixes. Section 7 concludes our work.

2 Background

Before deep diving into the definitions and proofs, we explain the big picture, introduce some terminology, and further clarify the relation to prior work.

Notation. Throughout the paper, $\{0,1\}^n$ represents the set of all n-length bit strings and $\{0,1\}^*$ the set of all finite length bit strings, including the empty string ϵ. We write $x \leftarrow F$ (resp. $x \overset{\$}{\leftarrow} F$) to denote that x is the return value of a deterministic (resp. probabilistic) algorithm F, $x \leftarrow y$ to assign y to variable x, and $x \overset{\$}{\leftarrow} \mathcal{S}$ to assign an element from $\mathcal{S}$, chosen uniformly at random (unless specified otherwise), to x. When initializing an empty set $\mathcal{L}$, we write $\mathcal{L} \leftarrow \emptyset$. We use $\perp$ to denote any uninitialized variable, as well as the return value from an algorithm that executes incorrectly.

Security Games. We follow closely [6] in our approach to formalizing security games. The games are presented in code form and they offer the adversary a number of oracles that allow it to animate arbitrary executions of the target protocols, in which it can select participants from four sets of entities: servers, human users, tokens and clients. Human users are represented in the model by the PIN they use to configure FIDO2 authenticators. We adopt here the typical approach to modelling password-based primitives in the cryptographic literature: users are represented by their identity and an associated PIN sampled from some (low entropy) distribution. The oracles allow the adversary to emulate the process of a user inserting its PIN via a client: the adversary identifies the client and the user in the oracle call, and the game then executes the client code on the user's PIN. The actions of other parties are captured by running the code prescribed by the protocol on their internal states and adversary-provided inputs, if any: different oracles capture different phases in the protocol execution, where the output of an oracle typically signals some event or information transfer that is visible to the adversary, and therefore may be used by the adversary in launching the next phase of its attack.

The games keep complex state to guarantee an accurate model of reality, and also to keep track of whether the adversary succeeded in breaking the protocol. In authentication games, breaking the protocol means that the adversary activated a winning condition: it caused some party to accept a message that allows the adversary to impersonate another party. In privacy games, breaking the protocol means that the adversary is able to link two protocol executions that should appear unrelated given the adversary's knowledge of the internal states of parties.

PlA and PACA. Barbosa et al. [3] defined the syntax and authentication security of FIDO2 by introducing two primitives called *Passwordless Authentication (PlA)* and *PIN-Based Access Control for Authenticators (PACA)*. WebAuthn is cast as an instance of PlA, and CTAP as an instance of PACA. This terminology is also adopted by Bindel et al. [6], where the authors refined the syntax and security models to more closely reflect the structure (and various versions) of the FIDO2 specification. Bindel et al. [6] focused on settings where no long-term attestation keys are used and hence assumed passive attackers in registration flows. Later, Bindel et al. [7] continued the work of [6], focusing only on WebAuthn, but considered various attestation modes and extended the analysis to privacy guarantees. More detailed background is summarized next.

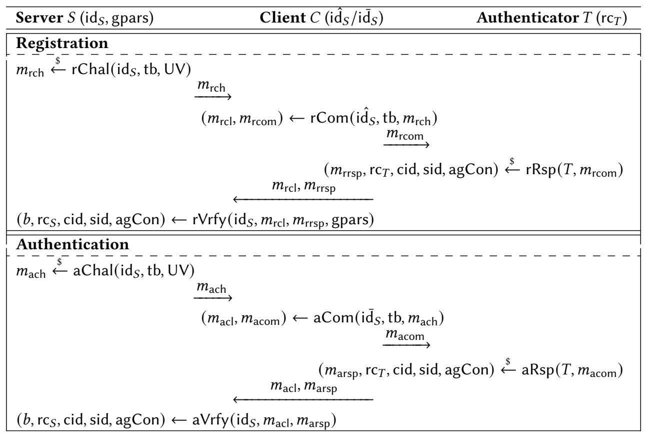

Server S $(\mathsf{id}_S, \mathsf{gpars})$
Client C $(\hat{\mathsf{id}}_S/\bar{\mathsf{id}}_S)$
Authenticator T (rc_T)
Registration
$m_{\mathrm{rch}} \overset{\$}{\leftarrow} \mathsf{rChal}(\mathsf{id}_S, \mathsf{tb}, \mathsf{UV})$
$\xrightarrow{m_{\mathrm{rch}}}$
$(m_{\mathrm{rcl}}, m_{\mathrm{rcom}}) \leftarrow \mathsf{rCom}(\hat{\mathsf{id}}_S, \mathsf{tb}, m_{\mathrm{rch}})$
$\xrightarrow{m_{\mathrm{rcom}}}$
$(m_{\mathrm{rrsp}}, \mathsf{rc}_T, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \overset{\$}{\leftarrow} \mathsf{rRsp}(T, m_{\mathrm{rcom}})$
$\xleftarrow{m_{\mathrm{rcl}}, m_{\mathrm{rrsp}}}$
$(b, \mathsf{rc}_S, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \leftarrow \mathsf{rVrfy}(\mathsf{id}_S, m_{\mathrm{rcl}}, m_{\mathrm{rrsp}}, \mathsf{gpars})$
Authentication
$m_{\mathrm{ach}} \overset{\$}{\leftarrow} \mathsf{aChal}(\mathsf{id}_S, \mathsf{tb}, \mathsf{UV})$
$\xrightarrow{m_{\mathrm{ach}}}$
$(m_{\mathrm{acl}}, m_{\mathrm{acom}}) \leftarrow \mathsf{aCom}(\bar{\mathsf{id}}_S, \mathsf{tb}, m_{\mathrm{ach}})$
$\xrightarrow{m_{\mathrm{acom}}}$
$(m_{\mathrm{arsp}}, \mathsf{rc}_T, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \overset{\$}{\leftarrow} \mathsf{aRsp}(T, m_{\mathrm{acom}})$
$\xleftarrow{m_{\mathrm{acl}}, m_{\mathrm{arsp}}}$
$(b, \mathsf{rc}_S, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \leftarrow \mathsf{aVrfy}(\mathsf{id}_S, m_{\mathrm{acl}}, m_{\mathrm{arsp}})$

Figure 2: The WebAuthn protocol (see Appendix A for complete descriptions of the algorithms). For attestation mode Basic**, attestation material is generated by the group initialization algorithm** $(\mathsf{gpars}, \mathsf{rc}) \overset{\$}{\leftarrow} \mathsf{GInit}$**, which for our purposes can be thought of as a signature key generation algorithm where** gpars **contains the attestation verification key** vk**, c.f., Appendix G). The token's registration context** rc_T **is initialized as** rc**, which contains the attestation private key** ak**; the server inputs its identity** $\mathbf{id}_S$ **and attestation info** gpars**. For attestation modes** None **and** Self**, both** rc_T **and** gpars **are empty.**

WebAuthn and PlA. We depict the WebAuthn protocol (with attestation mode Basic) in Figure 2 by following the descriptions in [6, 7]. As in [7], our WebAuthn description captures the cryptographic core of both the current stable version WebAuthn 2 [24] and a working draft for WebAuthn 3 [25], so we henceforth simply refer to this protocol as WebAuthn[4].

WebAuthn has two challenge-response flows, one for registering a new credential at the server (with algorithms rChal, rCom, rRsp, and rVrfy) and one for authenticating under a previously registered credential (with algorithms aChal, aCom, aRsp, and aVrfy).

On registration, the authenticator (token) generates a new key pair (pk, sk) and, unless the attestation mode is None, it signs the new credential pk, received random challenge and other relevant metadata. When using attestation mode Basic, the attestation parameters gpars allow the server to verify the signature and ensure that pk was indeed generated by a secure hardware device; depending on the application, the server may have different ways to validate the attestation public keys vk used to verify the above attestation signatures but here, for simplicity, we assume the server can extract the valid vk from the input gpars. In attestation mode Self, the attestation signing key is the credential's associated private key sk itself, so this mode can be viewed as the same as None with an extra proof by the token that it knows the signing key for the freshly generated credential. In an authentication run, the process is similar, but the token now always uses sk to sign the challenge and metadata. Note that, in cases where the client can verify the identity of the server (e.g., through a TLS connection), the user can rely on the client to abort any WebAuthn runs where the metadata sent to the token does not encode the correct intended server identity $\hat{\mathsf{id}}_S/\bar{\mathsf{id}}_S$.

The PlA authentication security models in [3, 6, 7] capture the following server-side guarantee of WebAuthn: if a server instance accepts an authentication run, then it is uniquely partnered with a token instance. Crucially, *partnership* in this setting guarantees that the response issued by the token can be linked back to a unique registration flow between the same server and token. In practice, the guarantee that a registered credential was indeed generated in a physical authenticator is achieved only if tokens can store long-term attestation keys.

We adopt the PlA authentication model of [7] with small changes: we simplify the model by removing attestation modes that are not commonly used in FIDO2 tokens, and we slightly modify the winning conditions for the adversary in order to clarify the overall guarantees provided by FIDO2. More precisely, we take the partnership definition from [3], which is more restrictive and therefore makes the model stronger. The details are in Appendix G, where we discuss the model differences to [7] and explain why the proofs in [7] actually still apply to our stronger model. We do not claim novelty at the PlA level, except for clarifying the security experiments and the implications of prior proofs.

CTAP 2.1 and PACA. The structure of the CTAP 2.1 protocol [1] is shown in Figure 3 (with details in Appendix B). Here, we outline CTAP 2.1 instantiated with the so-called PIN/UV Auth Protocol 2, denoted by puvProtocol, and omit the other PIN/UV Auth Protocol 1 instantiation that is essentially the legacy protocol in a previous version CTAP 2.0 [13]. Their core differences are highlighted in Appendix B, where we present the details of CTAP protocols.

With CTAP 2.1, a client can set up, bind to, and unlock an authenticator (token) with a user PIN (pin). In addition to the possibility that the token may be rebooted and then regenerate some ephemeral cryptographic parameters, the protocol has two phases.

The Setup phase is typically executed once and is independent of WebAuthn operations; it is used to configure the token with a hash of the user pin. More precisely, the client first obtains some information about the token's capabilities and informs it that Setup is about to start; the token then returns a DH share pk_T. The client completes Setup by sending its own DH share c along with an authenticated encryption (c_p, t_p) of the user pin. The token stores only the hash of the pin, named pinHash.

[4]WebAuthn 3 [25] introduces new features to enhance usability (e.g., support for cross-origin iFrames), expand API functionality (e.g., support for passkey authenticators), etc., but its cryptographic core (as abstracted in our work and [7]) is not changed.

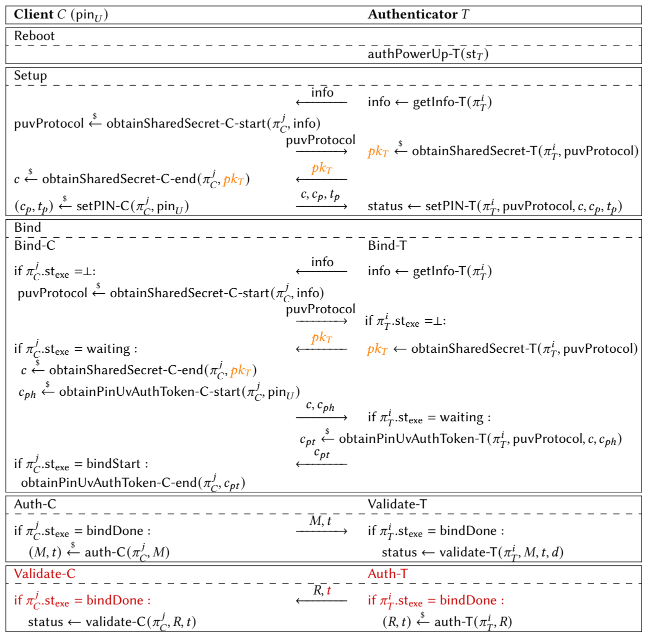

Client C (pin_U)
Authenticator T
Reboot
$\text{authPowerUp-T}(\text{st}_T)$
Setup
info
$\text{info} \leftarrow \text{getInfo-T}(\pi_T^i)$
$\text{puvProtocol} \xleftarrow{\$} \text{obtainSharedSecret-C-start}(\pi_C^j, \text{info})$
puvProtocol
$pk_T \xleftarrow{\$} \text{obtainSharedSecret-T}(\pi_T^i, \text{puvProtocol})$
pk_T
$c \xleftarrow{\$} \text{obtainSharedSecret-C-end}(\pi_C^j, pk_T)$
c, c_p, t_p
$(c_p, t_p) \xleftarrow{\$} \text{setPIN-C}(\pi_C^j, \text{pin}_U)$
$\text{status} \leftarrow \text{setPIN-T}(\pi_T^i, \text{puvProtocol}, c, c_p, t_p)$
Bind
Bind-C
Bind-T
info
if $\pi_C^j.\text{st}_{\text{exe}} = \perp$:
$\text{info} \leftarrow \text{getInfo-T}(\pi_T^i)$
$\text{puvProtocol} \xleftarrow{\$} \text{obtainSharedSecret-C-start}(\pi_C^j, \text{info})$
puvProtocol
if $\pi_T^i.\text{st}_{\text{exe}} = \perp$:
pk_T
if $\pi_C^j.\text{st}_{\text{exe}} = \text{waiting}$:
$pk_T \leftarrow \text{obtainSharedSecret-T}(\pi_T^i, \text{puvProtocol})$
$c \xleftarrow{\$} \text{obtainSharedSecret-C-end}(\pi_C^j, pk_T)$
$c_{ph} \xleftarrow{\$} \text{obtainPinUvAuthToken-C-start}(\pi_C^j, \text{pin}_U)$
c, c_{ph}
if $\pi_T^i.\text{st}_{\text{exe}} = \text{waiting}$:
$c_{pt} \xleftarrow{\$} \text{obtainPinUvAuthToken-T}(\pi_T^i, \text{puvProtocol}, c, c_{ph})$
c_{pt}
if $\pi_C^j.\text{st}_{\text{exe}} = \text{bindStart}$:
$\text{obtainPinUvAuthToken-C-end}(\pi_C^j, c_{pt})$
Auth-C
Validate-T
M, t
if $\pi_C^j.\text{st}_{\text{exe}} = \text{bindDone}$:
if $\pi_T^i.\text{st}_{\text{exe}} = \text{bindDone}$:
$(M, t) \xleftarrow{\$} \text{auth-C}(\pi_C^j, M)$
$\text{status} \leftarrow \text{validate-T}(\pi_T^i, M, t, d)$
Validate-C
Auth-T
R, t
if $\pi_C^j.\text{st}_{\text{exe}} = \text{bindDone}$:
if $\pi_T^i.\text{st}_{\text{exe}} = \text{bindDone}$:
$\text{status} \leftarrow \text{validate-C}(\pi_C^j, R, t)$
$(R, t) \xleftarrow{\$} \text{auth-T}(\pi_T^i, R)$

Figure 3: The CTAP 2.1 protocol (black), our fixed CTAP 2.1+ protocol (with red-colored block), and our fixed CTAP 2.1++ protocol (where the token's DH share pk_T is *regenerated* in every obtainSharedSecret-T execution). Complete descriptions of the algorithms are shown in Appendix B.

When the user wants to register a new credential or authenticate to a server using an existing one, the client executes a sequence of Bind, Auth-C and Validate-T subprotocols. Bind starts with a Diffie-Hellman key exchange as the one performed in Setup (where the token might reuse its DH share generated from Setup but the client always regenerates a fresh DH share); then, rather than transmitting an authenticated encryption of the pin, the client transmits an unauthenticated CBC encryption c_{ph} of the pinHash. The token completes Bind by transmitting back a CBC encryption of pt, the so-called pinToken[5], using the *same* symmetric key that encrypted the pinHash. The pinToken is simply a fresh MAC key that can be used by the client in Auth-C to send authorized commands to the token, which then validates them in Validate-T. These commands are WebAuthn requests (challenges) for registration or authentication operations. Token responses are sent back to the client, but *not* cryptographically protected at the CTAP level.

The PACA authentication security model in [3] assumes the adversary to be passive during Setup, as there is no prior common context between the client and token. This trust assumption is adopted by all subsequent works. Furthermore, the adversary is *not* allowed to actively attack the client during Bind, since CTAP uses unauthenticated Diffie-Hellman key exchange. However, the adversary is allowed to actively try to establish new bindings to the token. Additionally, the adversary can ask client instances that completed a binding to authorize commands of its choice, and its goal is to forge one such authorized command. A forgery here is defined as having a token instance accept a command that was not transmitted by its unique binding partner (a client instance).

The PACA model given in [6] refines the one in [3] in a number of ways, in order to more closely capture CTAP 2.1 operations. Furthermore, their model was strengthened such that it allows a limited active attack on clients during Bind: in the concrete case of the CTAP 2.1 protocol, this means that the attacker now has the power to control the *last* message sent from the token back to the client, which contains the encrypted pinToken. We show that this strengthened model gives too much power to the adversary, by identifying an attack that renders CTAP 2.1 insecure in such a model, invalidating the security claim made in [6]. The attack and our fix are discussed in detail in Section 4.2.

Composing PlA and PACA. The authentication security of FIDO2 as a whole is captured in [3, 6] by a composed model that considers the joint operation of PlA and PACA protocols. In this model, the adversary has access to all the PACA oracles that model Setup and Bind. Additionally, it can run PlA+PACA challenge-response interactions that involve client and token instances that it may have set up in arbitrary ways: formally, the PlA registration and authentication oracles are replaced by PlA+PACA oracles that model the need for authorizing and validating WebAuthn requests (and in this work also WebAuthn responses) with CTAP operations in the full FIDO2. The goal of the adversary is to break the guarantees we described in our authentication results (second bullet) in the Introduction. Results in [3, 6] show that the only way to break these guarantees is to break either the underlying PlA or PACA primitive.

Crucially, the above models assume that the attacker has the power to feed the server arbitrary responses to its challenges during authentication (and registration) runs, which reflects the choice of FIDO2 to confer no protection to responses when they are transmitted from tokens back to clients. However, neither of them captures security against rogue-key attacks.The model in [6] focuses on attestation mode None, so it explicitly disallows active attacks during registration and calls this assumption *trust on first use*. When attestation mode Basic is in use, the model in the full version of [3] allows active attacks in registration, but it cannot prevent rogue key attacks since the attacker can successfully register with a valid token from the same batch as the user's token.

In the next section we specify the syntax for *PIN-based access control for authenticators (PACA)* protocols that capture CTAP, and refer to Appendix G for the syntax and security models for *passwordless authentication (PlA)* protocols.

3 (m)PACA Protocol Syntax

Our PACA syntax closely follows [6], which itself follows [3]. To capture our authentication fix of CTAP, we later extend PACA to an mPACA protocol, where the leading "m" stands for "mutually authenticated".

A PACA protocol is an interactive protocol between 3 parties: a client C, an authenticator (or token) T and a user U. It comprises five subprotocols: Reboot, Setup, Bind, Auth-C and Validate-T. Below, when we say a protocol takes as input C and/or T, we mean the states of them, which are updated during the protocol execution.

Reboot: inputs a token T and initializes (or refreshes) its state, to prepare for the execution of the other PACA subprotocols.

[5]In CTAP 2.1, the full name of pt is pinUvAuthToken, which we simply call pinToken.

This subprotocol powers up the token and always requires *user interaction* (e.g., a USB token being plugged out and then plugged in to a machine).

Setup: inputs a token T, client C, and user U who participates by providing a pin to C. During this subprotocol execution, C securely transmits pin to T, then T saves information about this pin (e.g., a hash thereof) in its static storage (i.e., not affected by Reboot) such that later it can be used by U to authorize clients access to T. This subprotocol is typically executed *once* per token.

Bind: inputs a token T, client C, and user U who again provides C with a pin. This subprotocol aims to create an authenticated channel from C to T. It can be expressed as multiple runs of client-side and token-side processing Bind-C and Bind-T. Bind-C inputs C, U and a message m and outputs a message m'. Bind-T inputs T and a message m and outputs a message m'. Depending on the stage of execution of Bind-C or Bind-T, m and m' will represent different types of information.

Auth-C: inputs a client C, command M, and outputs (M, t), where t is a tag that authorizes command M. This subprotocol authorizes commands that C sends to token T (using the authenticated channel established from Bind).

Validate-T: inputs a token T, command M, tag t and user decision bit d. This subprotocol validates command M given tag t and user decision bit d, and outputs a bit as the result of the validation.

We extend the PACA syntax with a function $\mathsf{Public}(T)$ that models the information a user or an attacker can learn about the current public state of token T (e.g., the token version). This information must be properly defined for the analyzed protocol.

Correctness imposes that a client-authorized command is accepted by the token if and only if a user approves the command ($d = 1$); the formal definition is essentially the same as that in [3] and omitted here.

Syntax for mPACA protocols. An mPACA protocol extends the syntax of PACA with two additional subprotocols:

Auth-T: inputs a token T, command M, and outputs (M, t), where t is a tag that authorizes command M. This subprotocol authorizes responses that T sends to client C (using the authenticated channel established from Bind).

Validate-C: inputs a client C, command M and tag t. This subprotocol validates command M given tag t and outputs a bit as the result of the validation.

Correctness is extended to further impose that a token-authorized command is accepted by the client.

Session oracles and states. We consider two types of session oracles π_T^i and π_C^j to specify the i^{th} and j^{th} instance of token T and client C, respectively. An (m)PACA protocol implements certain states for client and tokens. Client session oracles are completely independent from each other and maintain no global state for any given C. Session oracles of token T each share a global state st_T, which contains the associated user identifier $\mathsf{st}_T.\mathsf{user}$, some information about the pin, and some initialization data $\mathsf{st}_T.\mathsf{initialData}$; the latter includes static configuration data like the supported protocol versions, and other protocol-specific states like a public counter that limits the maximum number of failed PIN tries. π_T^i and π_C^j have a binding state bs, session identifier sid and execution state $\mathsf{st}_{\mathsf{exe}} \in \{\perp, \mathsf{waiting}, \mathsf{bindStart}, \mathsf{bindDone}\}$. Here $\perp$ indicates that the session (oracle) is not yet initialized, in which case we simply write $\pi_T^i = \perp$ or $\pi_C^j = \perp$. When describing CTAP protocols as (m)PACA instances in this paper, following prior work, we use session oracles to simplify presentation and model the fact that an incoming message is processed in the context of a specific session.

4 Authentication Properties

We start with the goal of authentication, because this was the main focus of previous works and, as mentioned in the Introduction, our privacy analysis relies on the authentication results.

4.1 (m)PACA Authentication Model

We closely follow [6] to define our authentication security model for PACA protocols, and extend it to capture mPACA security. As presented in Figures 4, 5, the authentication security of a PACA protocol PACA is defined with a security experiment $\mathsf{Expt}_{\mathsf{PACA}}^{\mathsf{SUF\text{-}t}}$ executed between a challenger and an adversary $\mathcal{A}$. The security notion is called *strong unforgeability with trusted-binding (SUF-t)*, which ensures that a token can only accept a command that was authorized by a client bound to the token under user permission.

Trust model. The model assumes a fully authenticated channel for all communications between clients and tokens during Setup. It also assumes that no active attacks can be carried out against any client during the whole execution of Bind, while fully active attacks on tokens are allowed. As stated in the Background, this is the same assumption made in [3], which differs from the model in [6] that allows for stronger adversaries that may carry out active attacks against clients at the end of Bind. We later expand on the impossibility of achieving this stronger security in Section 4.2.

Experiment-specific boolean variables. Each session (oracle) has a isValid variable stating if this session is available for interaction with the adversary. Additionally, each token session has a $\pi_T^i.\mathsf{pinCorr}$ variable determining if the pin associated with token T has been corrupted, and each client session has a $\pi_C^j.\mathsf{compromised}$ variable determining if its internal state has been compromised.

Experiment oracles. We showcase the $\mathsf{Expt}_{\mathsf{PACA}}^{\mathsf{SUF\text{-}t}}$ oracles in Figure 5, which closely follow the code-based description in [6]. NewT and NewU generate new tokens and new users, respectively. In this model, a user is simply a holder of a pin, which can then be used to run Setup with multiple tokens. CorruptUser permanently corrupts a user U (by flagging it as corrupt) and returns his/her pin to $\mathcal{A}$. The Setup oracle performs a full Setup subprotocol run between two sessions π_C^j and π_T^i with user U's pin, and returns the trace of communications to $\mathcal{A}$. Likewise, Execute performs a full Bind subprotocol run between two sessions π_C^j and π_T^i, using the pin that was stored in T, invalidating all of T's previous sessions; the full trace of communications is then given to $\mathcal{A}$. Compromise returns a client session's binding state $\pi_C^j.\mathsf{bs}$ and permanently marks this session as compromised. Reboot on a token session π_T^i calls Reboot subprotocol on T and marks all of T's sessions as invalid. Auth-C authorizes a message M on a client session π_C^j using its binding state and outputs the same message and a tag t. Validate-T inputs a message M, tag t and user decision bit d on a token session π_T^i and outputs a boolean response. Finally, Send-Bind-T allows $\mathcal{A}$ to send a message m to a token session π_T^i to initiate, continue

```
tokenBindPartner $(T, i)$:
 1: if $\exists(C, j)$ s.t. $\pi_T^i$.sid $= \pi_C^j$.sid then
 2:     return $(C, j)$
 3: return $(\bot, \bot)$

Expt$^{\text{SUF-t}}_{\text{PACA}}$ $(\mathcal{A})$:
 1: $\mathcal{L}_{\text{authC}} \leftarrow \emptyset$
 2: win-SUF-t $\leftarrow 0$
 3: $() \xleftarrow{\$} \mathcal{A}^{O}(1^\lambda)$
 4: return win-SUF-t

Token-Win-SUF-t $(T, i, M, t, d)$:
 1: if $d \neq$ accepted then return 1
 2:
 3: if $\exists(C_1, j_1), (C_2, j_2)$
      $s.t.$ $(C_1, j_1) \neq (C_2, j_2)$ and $\pi_{C_1}^{j_1}$.st$_{\text{exe}}$ =
      $\pi_{C_2}^{j_2}$.st$_{\text{exe}}$ = bindDone
      and $\pi_{C_1}^{j_1}$.sid $= \pi_{C_2}^{j_2}$.sid then return 1
 4:
 5: if $\exists(T_1, i_1), (T_2, i_2)$ $s.t.$ $(T_1, i_1) \neq$
      $(T_2, i_2)$ and $\pi_{T_1}^{i_1}$.st$_{\text{exe}}$ = $\pi_{T_2}^{i_2}$.st$_{\text{exe}}$ =
      bindDone
      and $\pi_{T_1}^{i_1}$.sid $= \pi_{T_2}^{i_2}$.sid then return 1
 6:
 7: $(C, j) \leftarrow$ tokenBindPartner$(T, i)$
 8: if $(C, j, M, t) \notin \mathcal{L}_{\text{authC}}$ then
 9:     if $(C, j) = (\bot, \bot)$ or
        $\pi_C^j$.compromised = false then
10:         if $\pi_T^i$.pinCorr = false then
11:             return 1
12: return 0
```

```
clientBindPartner $(C, j)$:
 1: if $\exists(T, i)$ s.t. $\pi_C^j$.sid $= \pi_T^i$.sid then
 2:     return $(T, i)$
 3: return $(\bot, \bot)$

Expt$^{\text{SUF-t}}_{\text{mPACA}}$ $(\mathcal{A})$:
 1: $\mathcal{L}_{\text{authC}}, \mathcal{L}_{\text{authT}} \leftarrow \emptyset$
 2: win-SUF-t $\leftarrow 0$
 3: $() \xleftarrow{\$} \mathcal{A}^{O}(1^\lambda)$
 4: return win-SUF-t

Client-Win-SUF-t $(C, j, M, t)$:
 1: if $\exists(C_1, j_1), (C_2, j_2)$
      $s.t.$ $(C_1, j_1) \neq (C_2, j_2)$ and $\pi_{C_1}^{j_1}$.st$_{\text{exe}}$ =
      $\pi_{C_2}^{j_2}$.st$_{\text{exe}}$ = bindDone
      and $\pi_{C_1}^{j_1}$.sid $= \pi_{C_2}^{j_2}$.sid then return 1
 2:
 3: if $\exists(T_1, i_1), (T_2, i_2)$
      $s.t.$ $(T_1, i_1) \neq (T_2, i_2)$ and $\pi_{T_1}^{i_1}$.st$_{\text{exe}}$ =
      $\pi_{T_2}^{i_2}$.st$_{\text{exe}}$ = bindDone
      and $\pi_{T_1}^{i_1}$.sid $= \pi_{T_2}^{i_2}$.sid then return 1
 4:
 5: $(T, i) \leftarrow$ clientBindPartner$(C, j)$
 6: if $(T, i, M, t) \notin \mathcal{L}_{\text{authT}}$ then
 7:     if $\pi_C^j$.compromised = false then
 8:         if $\pi_T^i$.pinCorr = false then
 9:             return 1
10: return 0
```

Figure 4: (m)PACA authentication security experiments and winning conditions. Code in red only for the mPACA model. O denotes all oracles available to $\mathcal{A}$, as shown in Figure 5. The winning conditions are checked in Validate-T and Validate-C.

or complete Bind; in the latter case, it also invalidates all of T's previous sessions, including π_T^i if the query caused T to reboot.

Session partnership. We say sessions π_C^j and π_T^i are *partners* if, and only if, they completed Bind and agree on the session identifier sid. The concrete instantiation of sid depends on the concrete protocol to be analyzed. In our analysis of CTAP 2.1, we take the sid to be the full trace of the Bind run.

Winning conditions. For a PACA protocol PACA, we say that an adversary $\mathcal{A}$ against the security experiment $\text{Expt}^{\text{SUF-t}}_{\text{PACA}}$ wins if it gets a message-tag pair (M, t) accepted by a token session π_T^i through the Validate-T oracle and one of the following holds: (i) the user decision bit $d \neq 1$, (ii) two token sessions complete Bind with the same sid, (iii) two client sessions complete Bind with the same sid, or (iv) T's pin was not corrupted and *either* π_T^i has no partner *or* its partner was not compromised and did not output (M, t). This is captured by the winning condition Token-Win-SUF-t in Figure 4, which is checked whenever the adversary queries Validate-T.

Security for mPACA protocols. The mPACA authentication model extends the PACA authentication model with two additional oracles Auth-T and Validate-C as shown in Figure 5 that provide the adversary with the power to observe token-authorized commands and check validity of such commands. The winning conditions for the mPACA security experiment $\text{Expt}^{\text{SUF-t}}_{\text{mPACA}}$ (shown in Figure 4) is also extended to reflect the additional attack vectors: an additional winning condition Client-Win-SUF-t is checked whenever the adversary queries Validate-C.

```
NewT $(T$, initialData$)$:
 1: if st$_T \neq \bot$ then return $\bot$
 2: st$_T$.initialData $\leftarrow$ initialData
 3: Reboot(st$_T$)
 4: return

CorruptUser $(U)$:
 1: if $\mathcal{L}_{\text{valid}}[U] = \bot$ then return $\bot$
 2: $\mathcal{L}_{\text{corrupt}}[U] \leftarrow$ true
 3: pin $\leftarrow \mathcal{L}_{\text{valid}}[U]$
 4: return pin

Reboot $(T)$:
 1: if st$_T = \bot$ then return $\bot$
 2: for all $i$ $s.t.$ $\pi_T^i \neq \bot$ do
 3:     $\pi_T^i$.isValid $\leftarrow$ false
 4: Reboot(st$_T$)
 5: return

Send-Bind-T $(T, i, m)$:
 1: if st$_T = \bot$ then return $\bot$
 2: if $\pi_T^i = \bot$ then
 3:     $\pi_T^i \leftarrow$ st$_T$
 4: if $\pi_T^i$.st$_{\text{exe}}$ = bindDone or $\pi_T^i$.isValid = false
      then return $\bot$
 5: $\pi_T^i$.pinCorr $\leftarrow \mathcal{L}_{\text{corrupt}}[$st$_T$.user$]$
 6: if $\pi_T^i$.st$_{\text{exe}}$ = waiting then
 7:     $m_T \leftarrow$ Bind-T$(\pi_T^i, m)$
 8:     $c_{pt}$ || calledReboot $\leftarrow m'$
 9:     if calledReboot = true then
10:         for all $i'$ $s.t.$ $\pi_T^{i'} \neq \bot$ do
11:             $\pi_T^{i'}$.isValid $\leftarrow$ false
12:     else if $\pi_T^i$.st$_{\text{exe}}$ = bindDone then
13:         for all $i' \neq i$ and $\pi_T^{i'} \neq \bot$ do
14:             $\pi_T^{i'}$.isValid $\leftarrow$ false
15: else
16:     $m_T \leftarrow$ Bind-T$(\pi_T^i, m)$
17: $\mathcal{L}_{\text{ch}}^{\text{bd}} \xleftarrow{\cup} \{(T, i)\}$
18: return $m_T$

Auth-C $(C, j, M)$:
 1: if $\pi_C^j = \bot$ or $\pi_C^j$.st$_{\text{exe}} \neq$ bindDone or
      $\pi_C^j$.isValid = false then return $\bot$
 2: $(M, t) \xleftarrow{\$}$ auth-C$(\pi_C^j, M)$
 3: $\mathcal{L}_{\text{authC}} \xleftarrow{\cup} \{(C, j, M, t)\}$
 4: $\mathcal{L}_{\text{ch}}^{\text{op}} \xleftarrow{\cup} \{(C, j)\}$
 5: return $(M, t)$

Auth-T $(T, i, M)$:
 1: if $\pi_T^i = \bot$ or $\pi_T^i$.st$_{\text{exe}} \neq$ bindDone or
      $\pi_T^i$.isValid = false then return $\bot$
 2: $(M, t) \xleftarrow{\$}$ auth-T$(\pi_T^i, M)$
 3: $\mathcal{L}_{\text{authT}} \xleftarrow{\cup} \{(T, i, M, t)\}$
 4: $\mathcal{L}_{\text{ch}}^{\text{op}} \xleftarrow{\cup} \{(T, i)\}$
 5: return $(M, t)$
```

```
NewU $(U)$:
 1: if $\mathcal{L}_{\text{valid}}[U] = \bot$ then
 2:     pin $\xleftarrow{\mathcal{D}} \mathcal{P}$
 3:     $\mathcal{L}_{\text{valid}}[U] \leftarrow$ pin
 4:     $\mathcal{L}_{\text{corrupt}}[U] \leftarrow$ false
 5: return

Setup $(T, i, C, j, U)$:
 1: pin $\leftarrow \mathcal{L}_{\text{valid}}[U]$
 2: if st$_T = \bot$ or $\pi_T^i \neq \bot$ or $\pi_C^j \neq \bot$ or pin $= \bot$ then
      return $\bot$
 3: $\pi_T^i \leftarrow$ st$_T$
 4: trans $\xleftarrow{\$}$ Setup$(\pi_T^i, \pi_C^j$, pin$)$
 5: $\pi_T^i$.isValid, $\pi_C^j$.isValid $\leftarrow$ false
 6: st$_T$.user $\leftarrow U$
 7: return trans

Compromise $(C, j)$:
 1: if $\pi_C^j = \bot$ or $\pi_C^j$.st$_{\text{exe}} \neq$ bindDone then return
      $\bot$
 2: $\pi_C^j$.compromised = True
 3: $\mathcal{L}_{\text{corr}} \xleftarrow{\cup} \{(C, j)\}$
 4: return $\pi_C^j$.bs

Execute $(T, i, C, j)$:
 1: pin $\leftarrow \mathcal{L}_{\text{valid}}[$st$_T$.user$]$
 2: if st$_T = \bot$ or $\pi_T^i \neq \bot$ or $\pi_C^j \neq \bot$ or pin $= \bot$ then
      return $\bot$
 3: $\pi_T^i \leftarrow$ st$_T$
 4: trans, $m_C, m_T \leftarrow \bot$
 5: while $\pi_C^j$.st$_{\text{exe}} \neq$ bindDone do
 6:     $m_T \xleftarrow{\$}$ Bind-T$(\pi_T^i, m_C)$
 7:     $m_C \xleftarrow{\$}$ Bind-C$(\pi_C^j, U, m_T)$
 8:     trans $\leftarrow$ trans || $m_T$ || $m_C$
 9: for all $i' \neq i$ and $\pi_T^{i'} \neq \bot$ do
10:     $\pi_T^{i'}$.isValid $\leftarrow$ false
11: $\mathcal{L}_{\text{ch}}^{\text{bd}} \xleftarrow{\cup} \{(T, i), (C, j)\}$
12: return trans

Validate-T $(T, i, M, t, d)$:
 1: if $\pi_T^i = \bot$ or $\pi_T^i$.st$_{\text{exe}} \neq$ bindDone or
      $\pi_T^i$.isValid = false then return $\bot$
 2: status $\leftarrow$ validate-T$(\pi_T^i, M, t, d)$
 3: if status = accepted then win-SUF-t $\leftarrow$
      Token-Win-SUF-t$(T, i, M, t, d)$
 4: $\mathcal{L}_{\text{ch}}^{\text{op}} \xleftarrow{\cup} \{(T, i)\}$
 5: return status

Validate-C $(C, j, M, t)$:
 1: if $\pi_C^j = \bot$ or $\pi_C^j$.st$_{\text{exe}} \neq$ bindDone or
      $\pi_C^j$.isValid = false then return $\bot$
 2: status $\leftarrow$ validate-C$(\pi_C^j, M, t)$
 3: if status = accepted then win-mSUF-t $\leftarrow$
      Client-Win-SUF-t$(C, j, M, t)$
 4: $\mathcal{L}_{\text{ch}}^{\text{op}} \xleftarrow{\cup} \{(C, j)\}$
 5: return status
```

Figure 5: Oracles for (m)PACA security experiments. Changes from [6] in blue. Code in red only for mPACA introduced later. Code in teal only for privacy described in Figure 7.

Advantage measures. For protocol prot ∈ {PACA, mPACA}, the authentication advantage of $\mathcal{A}$ is defined as:

$$\text{Adv}^{\text{SUF-t}}_{\text{prot}}(\mathcal{A}) = \Pr[\text{Expt}^{\text{SUF-t}}_{\text{prot}}(\mathcal{A}) = 1] \ .$$

Differences to [6]. Our security model modifies the Execute and Send-Bind-T oracles from [6] back to the modeling approach of [3], in order to exclude active attacks on clients. This means that Execute captures a full Bind execution between two sessions π_C^j and π_T^i, whereas Send-Bind-T allows the adversary to arbitrarily attempt to bind with a token session π_T^i. We remove the Send-Bind-C oracle defined in [6], which further allows the adversary to behave actively when *completing* Bind on the client side.

Another minor difference is that we modify the Execute oracle's input, which no longer includes the user U. Therefore, Execute always uses the configured PIN information stored in the involved token. Not doing so, as in [6], implies that the adversary could always convince another user to interact with a token, and trivially win the game if the PINs coincide. This adds to the advantage of the adversary a non-negligible term that depends on the number of passive protocol executions, which is non-standard in the modeling of low-entropy secrets such as passwords and PINs.

4.2 Authentication Security of CTAP 2.1

While the proof in [6] gives undoubtedly a valuable and comprehensive security analysis of CTAP 2.1 that broadens the coverage and corrects some aspects in the initial proof given in [3], it still has some shortcomings that we fix in this paper. In the following, we first briefly discuss the main issue with the proof and its implications—with further details and proof fixes described in Appendix F—then show our security results.

Active attacks against clients during Bind. The problem lies in step of the proof of CTAP 2.1 [6, Appendix I, Game 16] that justifies removing PIN hashes from CBC encryptions during Bind. There, it is argued that CBC security guarantees that an attacker delivering a mauled ciphertext back to the client in the final step of the Bind subprotocol has no information on the resulting decrypted pinToken. We show that this is not true by presenting a simple attack, inspired by CBC padding oracle attacks [8]. Recall that the adversary obtains a CBC encryption c_{ph} of the pinHash in the last-but-one flow of Bind, and that the client is expecting to receive back a pinToken encrypted under the *same* symmetric key in the final flow. The attacker can therefore take the CBC encryption of the pinHash and echo it back to the client, who will recover a pinToken that encodes the *hash* of the user pin—see Appendix F for a detailed flow. When the client issues a command under this pinToken, the adversary immediately obtains enough information to perform an offline dictionary attack and recover the user pin.

We draw two conclusions from our attack: (i) the strongest model in which one can prove authentication security of CTAP 2.1 must assume the adversary to be passive against clients during Bind, and (ii) this gives further evidence that the Bind subprotocol of CTAP 2.1 should better be modified to withstand active attacks as recommended in [3]. As mentioned in the background Section 2, the current version of CTAP cannot be proved secure if the adversary can actively attack clients during Bind, because Bind uses *unauthenticated* Diffie-Hellman key exchange.[6] Nevertheless we note that, as noted in [9], man-in-the-middle atacks are much harder to launch over an USB channel than the final message injection required for a rogue-key attack. In what follows we will therefore propose a minimal change to CTAP that mitigates rogue-key injection only. We do not propose further patches to Bind to thwart more powerful man-in-the-middle attacks, and refer the interested readers to [3] for a (much more intrusive) solution to this problem based on a standard PAKE.

CTAP 2.1 security. Now, we state our authentication security result for the PACA protocol, CTAP 2.1 instantiated with PIN/UV Auth Protocol 2, which is described in Figure 3 and Appendix B. Though we do not formally analyze its PIN/UV Auth Protocol 1 instantiation here, in Appendix E we present how its security can be proved by adapting our proof for Theorem 1 shown below.

We assume that every user pin is sampled according to some distribution $\mathscr{D}$ with min-entropy $h_{\mathscr{D}}$ over the set $\mathcal{P}$ of all valid pins. We also assume that HKDF-SHA-256 [20] is modeled as a random oracle $\mathcal{H}_2$. Then, for other CTAP 2.1 building blocks, hash function H outputs the leftmost 128 bits of the SHA-256 digest, q is the prime order of the underlying elliptic-curve Diffie-Hellman (ECDH) group (for NIST curve P-256) [18], the underlying symmetric encryption scheme SKE denotes AES-256 in CBC mode with random IV [10], and the message authentication code MAC denotes HMAC-SHA-256 with 256-bit keys [5]. A query to Send-Bind-T on a session π_T^i is considered to be *active*, if $\mathcal{A}$ delivers a DH share c and ciphertext c_{ph} to π_T^i and (c, c_{ph}) was not output by any client session that was involved in a previous Execute query with a token session using the same token DH share. The following theorem states the SUF-t security of CTAP 2.1.

Theorem 1. *For every efficient adversary $\mathcal{A}$ that makes at most $q_S, q_E, q_{Send}, q_{NT}$ and q_R queries to Setup, Execute, Send-Bind-T, NewT and Reboot, and at most q_{Send}^{act} active queries to Send-Bind-T, there exist efficient adversaries $\mathcal{B}_1, \mathcal{B}_4, \mathcal{B}_5, \mathcal{B}_6$ and $\mathcal{B}_{10}$ such that:*

$$\begin{aligned} Adv_{CTAP\,2.1}^{SUF\text{-}t}(\mathcal{A}) \leq\ & q_{Send}^{act}/2^{h_{\mathscr{D}}} + (q_S + q_E + q_{Send})\ Adv_{ECDH}^{sCDH}(\mathcal{B}_1) \\ & + (q_S + q_E + q_{NT} + q_R + 2q_{Send})^2/(2q) + Adv_H^{coll}(\mathcal{B}_4) \\ & + q_S\ Adv_{SKE}^{IND\text{-}1\$PA\text{-}LHPC}(\mathcal{B}_5) + q_E\ Adv_{SKE}^{IND\text{-}1\$PA\text{-}LPC}(\mathcal{B}_6) \\ & + (q_E + q_{Send})^2/(2^{2\lambda+1}) + (q_E + q_{Send})\ Adv_{MAC}^{SUF\text{-}CMA}(\mathcal{B}_{10})\ . \end{aligned}$$

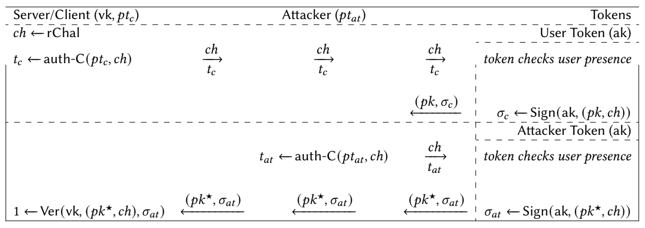
We provide security definitions of the underlying primitives used in the theorem in Appendix C, the proof sketch in Appendix D.1 and the full code-based proof in the full version [2].

4.3 Authentication Security of FIDO2

Recall that the latest version of FIDO2 consists of WebAuthn and CTAP 2.1. WebAuthn has been proved in [7] to offer PlA authentication security. We reuse these results with minor modifications, and the syntax and security model of PlA protocols in Appendix G. We now briefly discuss what we obtain in terms of composed security for FIDO2. These results follow along the lines of those outlined in [6], with natural adaptations due to our fixes to their CTAP 2.1 results. As we do not claim significant novelty for these results, we summarize the core ideas here and leave more details in Appendix I. Then, we explain rogue key attacks and how they affect the model.

Composed authentication of FIDO2. The composed model in which the current version of FIDO2 in attestation modes None and Self can be proved secure requires assuming *trust on first use (TOFU)*, i.e., the adversary is passive during the full registration run. Under this restriction, we can show that the server gets all the guarantees provided by PlA in these attestation modes. In attestation mode Basic, one can lift the TOFU assumption and guarantee the server that the accepted registered credentials is generated by valid tokens from the attested batch, as guaranteed by the PlA primitive. This is to be expected. But what does PACA provide, in addition to PlA security? As discussed at the end of Appendix I, PACA guarantees

[6]Using authenticated encryption during Bind would eliminate the specific attack we describe, but active attacks against clients in other part of Bind are still possible.



Server/Client (vk, pt_c)
Attacker (pt_{at})
Tokens
$ch \leftarrow$ rChal
User Token (ak)
$t_c \leftarrow$ auth-C(pt_c, ch)
$\xrightarrow[t_c]{ch}$
$\xrightarrow[t_c]{ch}$
$\xrightarrow[t_c]{ch}$
token checks user presence
$\xleftarrow{(pk, \sigma_c)}$
$\sigma_c \leftarrow$ Sign(ak, (pk, ch))
Attacker Token (ak)
$t_{at} \leftarrow$ auth-C(pt_{at}, ch)
$\xrightarrow[t_{at}]{ch}$
token checks user presence
$1 \leftarrow$ Ver(vk, ($pk^\star$, ch), σ_{at})
$\xleftarrow{(pk^\star, \sigma_{at})}$
$\xleftarrow{(pk^\star, \sigma_{at})}$
$\xleftarrow{(pk^\star, \sigma_{at})}$
$\sigma_{at} \leftarrow$ Sign(ak, ($pk^\star$, ch))

Figure 6: The rogue key attack (against a registration run) in Basic attestation. Right arrows represent commands for requesting a credential and left arrows are the respective responses. Procedures are simplified for better presentation.

that no attacker can authenticate under a credential stored in the user's token, unless it breaks the PACA access-control mechanism (modeled by compromising the client connected to the user's token), or simply corrupts the user pin.

Note here that the main differences to the result in [6] are that we further clarify what composed authentication security means for attestation modes None, Self and Basic, and that we highlight the need to assume that the composed model must still disallow active attacks against clients during Bind of the PACA protocol. At the end of Appendix I, we also remark that the authentication guarantee of FIDO2 can be strengthened if the server is looking a priori for a specific credential identifier that is associated with the authenticating user. Otherwise, although the attacker cannot impersonate the user due to the PACA protection captured by the composed model, it is still possible for the attacker to launch an attack similar to rogue key attacks, but here targeting an authentication run; this may allow an attacker to authenticate the user under a credential that is under the attacker's control. I.e., if the user is authenticating via an honest client, the best the attacker can do is to log the user in to an account that is under the attacker's control. Our proposed fixes to the protocol also eliminate this attack.

Rogue key attacks. In Figure 6 we recall the setting described in [4] in which an attacker manages to register a rogue key $pk^\star$ instead of the user's legitimate credential pk. To launch the attack in attestation modes None or Self, the attacker doesn't need any special equipment and can just create the rogue credential by itself. However, if using attestation mode Basic and the server has information that allows recognizing a target batch (or group), then the attacker must have access to its own token, from the same batch as the user's token (and hence sharing the same attestation private key ak). The figure shows this latter case.

In the top part of Figure 6, the server issues a challenge ch for the registration run, which is authorized with pinToken pt_c by the client and then sent to the user's token. The attacker can easily observe the challenge because it is not encrypted. If this attacker is able to inject a message that replaces the response sent by the user's token to the client, as described in [4, 9], then it can perform the actions shown in the bottom part of the figure: it takes the same challenge and uses its own token (using some other pinToken pt_{at} to access it) to generate and sign a new credential. This malicious credential will be accepted by the server, since the attacker's token is from the same batch as the user's token and uses the same ak.

Failure to capture rogue key attacks formally. The security model in [6] and our composed model sketched above do not capture rogue key attacks, because in the entire registration flow it is assumed that the adversary is passive. If this was not the case, then the rogue key attack would lead to a break of the protocol in these models because the messages sent by the token to the client are not authenticated (formally in Appendix H it is captured by the setting in which Validate-C always returns true). This means that an adversary can simply use its PACA oracles in the security experiment to gain control over some other token, use it to generate a credential for the challenge created by the target server, and feed it to the honest client who will pass it along to the server.

As a side note, we recall the observation in [4] that if the server knows a priori a unique attestation public key for the user's token (e.g., it is in a batch of size 1), as assumed in [3], then the adversary will never be able to find a token that allows launching the attack. So, the model could be strengthened to prevent rogue key attacks by restricting the batch size as 1, but this hypothesis is not realistic.

4.4 CTAP 2.1+ for Stronger Authentication

In this section, we capture rogue key attacks by removing the restriction that the adversary cannot be active during the registration run. Clearly, this security is unachievable for the current version of FIDO2, so we investigate how to fix it to resist rogue key attacks.

CTAP 2.1+ and its authentication security. We propose a modification to CTAP 2.1, the fixed protocol called CTAP 2.1+, that achieves SUF-t security in our stronger mPACA authentication model and, therefore, provides protection against rogue key attacks. Our CTAP 2.1+ modifications are highlighted in red in Figure 3, with detailed descriptions provided in Appendix B.

As with CTAP 2.1, CTAP 2.1+ still authorizes a client-to-token command M with Auth-C and validates a client-authorized command (M, t) with Validate-T; moreover, CTAP 2.1+ introduces two more algorithms Auth-T and Validate-C to authorize a token-to-client response R and validate a token-authorized response (R, t). Regarding low-level functions, our CTAP 2.1+ protocol stops using the pinToken provided by the token directly as a MAC key, but uses a key derivation function KDF to expand it to two MAC keys, and use them to authorize commands in both directions.

The following theorem states our result for CTAP 2.1+, with the above KDF modeled as a random oracle $\mathcal{H}_3$.

Theorem 2. *For every efficient adversary $\mathcal{A}$ that makes at most q_S, q_E, q_{Send}, q_{NT} and q_R queries to Setup, Execute, Send-Bind-T, NewT and Reboot, at most q^{act}_{Send} active queries to Send-Bind-T, and at most $q_{\mathcal{H}_3}$ queries to $\mathcal{H}_3$, there exist efficient adversaries $\mathcal{B}_1$, $\mathcal{B}_4$, $\mathcal{B}_5$, $\mathcal{B}_6$, $\mathcal{B}_{11}$ and $\mathcal{B}_{12}$ such that:*

$$\begin{aligned}
Adv^{SUF\text{-}t}_{CTAP\,2.1+}(\mathcal{A}) &\le q^{act}_{Send}/2^{h_{\mathcal{D}}} + (q_S + q_E + q_{Send})Adv^{sCDH}_{ECDH}(\mathcal{B}_1)\\
&+ (q_S + q_E + q_{NT} + q_R + 2q_{Send})^2/(2q) + Adv^{coll}_{H}(\mathcal{B}_4)\\
&+ q_S\, Adv^{IND\text{-}1\$PA\text{-}LHPC}_{SKE}(\mathcal{B}_5) + q_E\, Adv^{IND\text{-}1\$PA\text{-}LPC}_{SKE}(\mathcal{B}_6)\\
&+ (q_E + q_{Send})^2/(2^{2\lambda+1}) + q_{\mathcal{H}_3}(q_E + q_{Send})/2^{2\lambda}\\
&+ (q_E + q_{Send})\, Adv^{SUF\text{-}CMA}_{MAC}(\mathcal{B}_{11}) + q_E\, Adv^{SUF\text{-}CMA}_{MAC}(\mathcal{B}_{12})\ .
\end{aligned}$$

We provide the proof sketch in Appendix D.2 and the full code-based proof in the full version [2].

Composing CTAP 2.1+ and WebAuthn. The composed model in which we analyze the composition of an mPACA protocol with a PlA protocol is given in Appendix H. At the high level, the main differences to the model in [6] are that we consider active attackers during registration and that we formally cover, not only attestation mode None, but also Self and Basic. Recall that uncompromised CTAP 2.1+ client sessions will only accept messages transmitted by a unique token session to which they are bound and that, in the composed model, honest client sessions are assumed to be uniquely bound to a server session (e.g., via a TLS connection).

In summary, we state and prove two theorems in Appendix I that establish the following intuitive results:

- For attestation modes None and Self one must consider that registration is carried out via an honest client, as otherwise one gets no PlA guarantees. In particular, the server would not even be assured that the credential comes from a hardware token.
- For attestation mode Basic we can guarantee PlA security even if the server is interacting with a compromised client. However, rogue key attacks are still possible in this setting.
- For all attestation modes, if a token is interacting with an honest client, then the token will only reply to challenges if it is unlocked by a client or attacker using the correct user PIN.
- For all attestation modes, if an interaction with the server occurs via an honest client session (be it in authentication or registration), then the composed PlA+mPACA security guarantees that the server session connected to that honest client session will only ever accept a PlA session originating in the unique token session bound to that client session via mPACA. *This means in particular that we guarantee security against rogue key attacks.*

5 Privacy Properties

User privacy is an important security goal for FIDO2. Informally, privacy means that different registrations do not reveal whether they are linked to the same or different tokens, and hence users. In this section, we provide the first formal provable privacy analysis of CTAP 2.1 and FIDO2 as a whole, then propose a simple fix to achieve a natural stronger privacy guarantee.

5.1 (m)PACA Privacy Model

We start by defining PACA privacy, a property that guarantees a strong form of unlinkability between PACA sessions. The adversarial capabilities are modeled similarly to those in the model for PACA authentication and our privacy model inherits the same trust model, but here the adversarial goal is different. Figure 7 shows our privacy security experiment $\mathsf{Expt}^{\mathsf{priv}}_{\mathsf{PACA}}$, associated with an adversary $\mathcal{A}$ and a PACA protocol PACA. Our privacy definitions follow the style of PlA privacy definition presented in [7, 17] and recalled and adapted in Appendix G.3.

Experiment phases and oracles. The experiment $\mathsf{Expt}^{\mathsf{priv}}_{\mathsf{PACA}}$ has 3 phases. In Phase 1, we let the adversary freely interact with the oracles available in the PACA authentication experiment $\mathsf{Expt}^{\mathsf{SUF\text{-}t}}_{\mathsf{PACA}}$ (see Figure 5). This is captured via the $\mathcal{O}$ oracle notation. In the challenge phase Phase 2, the adversary specifies two (not necessarily distinct) uncorrupted "challenge" tokens T_0 and T_1, two clients C_0 and C_1, and two users U_0 and U_1. The challenger then calls InitRL, which will sample a random bit b and use (T_b, C_b) and (T_{1-b}, C_{1-b}) to respectively initialize oracles with suffixes LEFT and RIGHT. Finally, in Phase 3, the adversary can continue interacting with oracles in $\mathcal{O}$ as in Phase 1, but cannot create new tokens. However, it can query LEFT and RIGHT oracles. We note that it is necessary for the experiment to have challenge clients, in addition to challenge tokens: the state of clients, which may be leaked to the server, could also leak information that compromises privacy.

Winning conditions. In order to win the experiment, the adversary $\mathcal{A}$ has to guess the secret random bit b, meaning being able to distinguish the interaction between (T_0, C_0) and (T_1, C_1).

```
Expt^{(w)priv}_{PACA/mPACA}(A):
1: $\mathcal{L}_{\mathsf{corr}} \leftarrow \emptyset, \mathcal{L}^{\mathsf{bd}}_{\mathsf{ch}} \leftarrow \emptyset, \mathcal{L}^{\mathsf{bd}}_{\mathsf{lr}} \leftarrow \emptyset, \mathcal{L}^{\mathsf{op}}_{\mathsf{ch}} \leftarrow \emptyset, \mathcal{L}^{\mathsf{op}}_{\mathsf{lr}} \leftarrow \emptyset, \mathsf{pb} \leftarrow 1$, freshDH ← 1
2: $\mathsf{st}_1 \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(1^\lambda)$ // Phase 1
3: $T_0, T_1, C_0, C_1, U_0, U_1, \mathsf{st}_2 \leftarrow \mathcal{A}(1^\lambda, \mathsf{st}_1)$ // Phase 2
4: $b \leftarrow \mathsf{InitRL}(T_0, T_1, C_0, C_1, U_0, U_1)$
5: $\mathcal{O}' \leftarrow (\mathcal{O} \backslash \{\mathsf{NewT}\})$
6: $b' \leftarrow \mathcal{A}^{\mathcal{O}', \mathsf{LEFT}, \mathsf{RIGHT}}(1^\lambda, \mathsf{st}_2)$ // Phase 3
7: $r \xleftarrow{\$} \{0, 1\}$
8: if Check-priv-PACA()=1 then return $b = b'$
9: else return $r$

Check-priv-PACA():
1: for all $(T_{\mathsf{ch}}, i) \in \mathcal{L}^{\mathsf{bd}}_{\mathsf{ch}}, (T_{\mathsf{lr}}, j) \in \mathcal{L}^{\mathsf{bd}}_{\mathsf{lr}}$ do
2:     if $T_{\mathsf{ch}} = T_{\mathsf{lr}}$ and no Reboot is performed between $(T_{\mathsf{ch}}, i)$ and $(T_{\mathsf{lr}}, j)$ then
3:         freshDH ← 0
4: for two consecutive queries $q_n, q_m$ do
5:     if $q_n$ is LEFT/RIGHT query and $q_m$ is regular query, or vice versa and $\mathsf{Public}(T_0) \neq \mathsf{Public}(T_1)$ after $q_n$ then
6:         $\mathsf{pb} \leftarrow 0$
7: $\mathsf{Check} \leftarrow T_0.\mathsf{pinCorr} = \mathsf{false}$ and $T_1.\mathsf{pinCorr} = \mathsf{false}$ and $\mathcal{L}^{\mathsf{bd}}_{\mathsf{lr}} \cap \mathcal{L}_{\mathsf{corr}} = \emptyset$
8: $\mathsf{S} \leftarrow (\mathcal{L}^{\mathsf{bd}}_{\mathsf{ch}} \cap \mathcal{L}^{\mathsf{op}}_{\mathsf{lr}}) \cup (\mathcal{L}^{\mathsf{op}}_{\mathsf{ch}} \cap \mathcal{L}^{\mathsf{bd}}_{\mathsf{lr}}) \cup (\mathcal{L}^{\mathsf{bd}}_{\mathsf{ch}} \cap \mathcal{L}^{\mathsf{bd}}_{\mathsf{lr}})$
9: if $\mathsf{S} = \emptyset$ and Check and $\mathsf{pb} = 1$ and freshDH = 1 then return 1
10: else return 0

InitRL($T_0, T_1, C_0, C_1, U_0, U_1$):
1: $b \xleftarrow{\$} \{0, 1\}$
2: Initialize LEFT and RIGHT oracles
3: return $b$

Setup-LEFT_{$T_b, C_b, U_b$}$(i, j)$:
1: return $\mathsf{Setup}(T_b, i, C_b, j, U_b)$
Setup-RIGHT_{$T_{1-b}, C_{1-b}, U_{1-b}$}$(i, j)$:
1: return $\mathsf{Setup}(T_{1-b}, i, C_{1-b}, j, U_{1-b})$
Bind-LEFT_{$T_b, C_b$}$(i, j)$:
1: $\mathcal{L}^{\mathsf{bd}}_{\mathsf{lr}} \overset{\cup}{\leftarrow} \{(T_0, i), (C_0, j)\}, \mathcal{L}^{\mathsf{bd}}_{\mathsf{lr}} \overset{\cup}{\leftarrow} \{(T_1, i), (C_1, j)\}$
2: return $\mathsf{Execute}(T_b, i, C_b, j)$
Bind-RIGHT_{$T_{1-b}, C_{1-b}$}$(i, j)$:
1: $\mathcal{L}^{\mathsf{bd}}_{\mathsf{lr}} \overset{\cup}{\leftarrow} \{(T_0, i), (C_0, j)\}, \mathcal{L}^{\mathsf{bd}}_{\mathsf{lr}} \overset{\cup}{\leftarrow} \{(T_1, i), (C_1, j)\}$
2: return $\mathsf{Execute}(T_{1-b}, i, C_{1-b}, j)$

Send-LEFT_{$T_b$}$(i, M)$
1: $\mathcal{L}^{\mathsf{bd}}_{\mathsf{lr}} \overset{\cup}{\leftarrow} \{(T_0, i)\}, \mathcal{L}^{\mathsf{bd}}_{\mathsf{lr}} \overset{\cup}{\leftarrow} \{(T_1, i)\}$
2: return Send-Bind-T$(T_b, i, M)$
Send-RIGHT_{$T_{1-b}$}$(i, M)$
1: $\mathcal{L}^{\mathsf{bd}}_{\mathsf{lr}} \overset{\cup}{\leftarrow} \{(T_0, i)\}, \mathcal{L}^{\mathsf{bd}}_{\mathsf{lr}} \overset{\cup}{\leftarrow} \{(T_1, i)\}$
2: return Send-Bind-T$(T_{1-b}, i, M)$
Auth-C-LEFT_{$C_b$}$(j, M)$:
1: $\mathcal{L}^{\mathsf{op}}_{\mathsf{lr}} \overset{\cup}{\leftarrow} \{(C_0, j)\}, \mathcal{L}^{\mathsf{op}}_{\mathsf{lr}} \overset{\cup}{\leftarrow} \{(C_1, j)\}$
2: $(M, t) \leftarrow$ Auth-C$(C_b, j, M)$
3: return $(t, \pi^j_{C_b}.\mathsf{bs})$
Auth-C-RIGHT_{$C_{1-b}$}$(j, M)$:
1: $\mathcal{L}^{\mathsf{op}}_{\mathsf{lr}} \overset{\cup}{\leftarrow} \{(C_0, j)\}, \mathcal{L}^{\mathsf{op}}_{\mathsf{lr}} \overset{\cup}{\leftarrow} \{(C_1, j)\}$
2: $(M, t) \leftarrow$ Auth-C$(C_{1-b}, j, M)$
3: return $(t, \pi^j_{C_{1-b}}.\mathsf{bs})$
Validate-T-LEFT_{$T_b$}$(i, M, t, d)$:
1: $\mathcal{L}^{\mathsf{op}}_{\mathsf{lr}} \overset{\cup}{\leftarrow} \{(T_0, i)\}, \mathcal{L}^{\mathsf{op}}_{\mathsf{lr}} \overset{\cup}{\leftarrow} \{(T_1, i)\}$
2: $d \leftarrow$ Validate-T$(T_b, i, M, t, d)$.
3: return $(d, \pi^i_{T_b}.\mathsf{bs})$
Validate-T-RIGHT_{$T_{1-b}$}$(i, M, t, d)$:
1: $\mathcal{L}^{\mathsf{op}}_{\mathsf{lr}} \overset{\cup}{\leftarrow} \{(T_0, i)\}, \mathcal{L}^{\mathsf{op}}_{\mathsf{lr}} \overset{\cup}{\leftarrow} \{(T_1, i)\}$
2: $d \leftarrow$ Validate-T$(T_{1-b}, i, M, t, d)$.
3: return $(d, \pi^i_{T_{1-b}}.\mathsf{bs})$
Auth-T-LEFT_{$T_b$}$(i, M)$:
1: $\mathcal{L}^{\mathsf{op}}_{\mathsf{lr}} \overset{\cup}{\leftarrow} \{(T_0, i)\}, \mathcal{L}^{\mathsf{op}}_{\mathsf{lr}} \overset{\cup}{\leftarrow} \{(T_1, i)\}$
2: $(M, t) \leftarrow$ Auth-T$(T_b, i, M)$
3: return $(t, \pi^i_{T_b}.\mathsf{bs})$
Auth-T-RIGHT_{$T_{1-b}$}$(i, M)$:
1: $\mathcal{L}^{\mathsf{op}}_{\mathsf{lr}} \overset{\cup}{\leftarrow} \{(T_0, i)\}, \mathcal{L}^{\mathsf{op}}_{\mathsf{lr}} \overset{\cup}{\leftarrow} \{(T_1, i)\}$
2: $(M, t) \leftarrow$ Auth-T$(T_{1-b}, i, M)$
3: return $(t, \pi^i_{T_{1-b}}.\mathsf{bs})$
Validate-C-LEFT_{$C_b$}$(j, M, t, d)$:
1: $\mathcal{L}^{\mathsf{op}}_{\mathsf{lr}} \overset{\cup}{\leftarrow} \{(C_0, j)\}, \mathcal{L}^{\mathsf{op}}_{\mathsf{lr}} \overset{\cup}{\leftarrow} \{(C_1, j)\}$
2: $d \leftarrow$ Validate-C$(C_b, j, M, t, d)$.
3: return $(d, \pi^j_{C_b}.\mathsf{bs})$
Validate-C-RIGHT_{$C_{1-b}$}$(j, M, t, d)$:
1: $\mathcal{L}^{\mathsf{op}}_{\mathsf{lr}} \overset{\cup}{\leftarrow} \{(C_0, j)\}, \mathcal{L}^{\mathsf{op}}_{\mathsf{lr}} \overset{\cup}{\leftarrow} \{(C_1, j)\}$
2: $d \leftarrow$ Validate-C$(C_{1-b}, j, M, t, d)$
3: return $(d, \pi^j_{C_{1-b}}.\mathsf{bs})$
```

Figure 7: (m)PACA privacy security experiments. $\mathsf{Expt}^{\mathsf{priv}}_{\mathsf{PACA}}$ for PACA privacy (without red or blue parts), $\mathsf{Expt}^{\mathsf{wpriv}}_{\mathsf{PACA}}$ for PACA weak privacy (without red parts), $\mathsf{Expt}^{\mathsf{priv}}_{\mathsf{mPACA}}$ for mPACA privacy (without blue parts), $\mathsf{Expt}^{\mathsf{wpriv}}_{\mathsf{mPACA}}$ for mPACA weak privacy (with everything). $\mathcal{A}^{\mathcal{O}}$ indicates that the adversary $\mathcal{A}$ has access to all PACA authentication oracles denoted by $\mathcal{O}$ (defined in Figure 5).

To avoid trivial wins, we follow the PlA privacy models [7, 17] to perform context separation checks. Check-priv-PACA will return 1 if all checks are passed, the experiment then proceeds to return 1 if adversary guess the bit correctly, and 0 otherwise. If at least one check fails, Check-priv-PACA will return 0 and the experiment will ignore bit b' output by advasary, and output a random bit r. Now we get into the details of Check-priv-PACA. Intuitively, after a particular Bind run between sessions π_T^i and π_C^j, suppose that $\mathcal{A}$ could specify (T_0, C_0) as (T, C) and (T_1, C_1) as some pair such that $\pi_{T_1}^i$ and $\pi_{C_1}^j$ do not have a matching Bind state; then $\mathcal{A}$ could trivially identify (T_0, C_0) by asking the challenge clients to authorize a command on session j or asking the challenge tokens to validate an authorized command on session i, because only $\pi_{T_0}^i$ and $\pi_{C_0}^j$ have a binding state. Therefore, we require that, if $\mathcal{A}$ queries an Execute oracle, the involved sessions cannot be queried to a LEFT/RIGHT oracle for Auth-C or Validate-T; this also applies to the converse case. The above is captured by requiring $(\mathcal{L}_{\mathsf{ch}}^{\mathsf{bd}} \cap \mathcal{L}_{\mathsf{lr}}^{\mathsf{op}}) \cup (\mathcal{L}_{\mathsf{ch}}^{\mathsf{op}} \cap \mathcal{L}_{\mathsf{lr}}^{\mathsf{bd}})$ to be empty. Additionally, we require that $\mathcal{A}$ cannot query the same sessions via both regular oracles and LEFT/RIGHT oracles for Bind runs. Otherwise, $\mathcal{A}$ can, for instance, query regular Execute on token T_0 with some session index i, and later query Bind-LEFT with the same session index i. If Bind-LEFT uses T_0, the oracle will return failure, otherwise, it runs Bind and returns success. This is captured by requiring empty $\mathcal{L}_{\mathsf{ch}}^{\mathsf{bd}} \cap \mathcal{L}_{\mathsf{lr}}^{\mathsf{bd}}$.

Recall that each token T may have public information that is available upon request, e.g., token version, number of remaining allowed failed PIN retries, and whether T has been set up. This is captured by Public(T) defined in Section 3. To avoid trivial wins, we require Public(T_0) = Public(T_1) holds whenever $\mathcal{A}$ makes a LEFT/RIGHT query after a regular query, or vice versa. Furthermore, the Check flag ensures that challenge tokens are not corrupted and challenge client sessions are not compromised.

Privacy for mPACA protocols. The privacy model for mPACA protocols is the same, except that we extend the experiment oracles to include Auth-T and Validate-C, and extend the winning conditions accordingly. See $\mathsf{Expt}_{\mathsf{mPACA}}^{\mathsf{priv}}$ in Figure 7.

Advantage measures. For protocol prot ∈ {PACA, mPACA}, the privacy advantage of $\mathcal{A}$ is defined as:

$$\mathsf{Adv}_{\mathsf{prot}}^{\mathsf{priv}}(\mathcal{A}) = |2 \Pr[\mathsf{Expt}_{\mathsf{prot}}^{\mathsf{priv}}(\mathcal{A}) = 1] - 1| \ .$$

5.2 Privacy of CTAP 2.1 and CTAP 2.1+

Privacy attacks against CTAP 2.1 and CTAP 2.1+. We observe that neither CTAP 2.1 nor CTAP 2.1+ meets our privacy definitions. The reason is that tokens do not generate a *fresh* Diffie-Hellman share for each key exchange. Formally, in our model an adversary can observe repeated DH shares sent by a token to break privacy, e.g., the adversary first observes a token session in Phase 1 to collect its DH share used in Setup, then chooses this *same* token in Phase 2, and finally in Phase 3 checks if the trace returned by the Bind-LEFT oracle repeats the DH share or not. This attack is formalized in Appendix J, where we describe an efficient adversary $\mathcal{A}$ such that:

$$\mathsf{Adv}_{\mathsf{prot}}^{\mathsf{priv}}(\mathcal{A}) = 1 - 1/q \approx 1 - 2^{-2\lambda} \ ,$$

where prot ∈ {CTAP 2.1, CTAP 2.1+}, q is the order of the underlying ECDH group, and $\lambda = 128$.

For the above attack, we do not claim that it has significant practical implications. Nevertheless, our tests show that, when a USB token is used to register or authenticate to multiple accounts without rebooting (by remaining plugged-in and not putting the computer to sleep), the token will reuse its DH share. This can allow a malicious server (that can access the CTAP communication, e.g., via malware with low-privilege access installed on the computer [4]) to link accounts (perhaps for different servers) of the same user, especially when multiple users share the same machine (e.g., a corporate or public computer). As we will show shortly, this potential privacy *leak* can be easily fixed by enforcing DH shares to be always refreshed on the token.

Weak privacy of CTAP 2.1 and CTAP 2.1+. In order to analyze the privacy guarantees achieved by CTAP 2.1 and CTAP 2.1+, we define weak privacy notions for both PACA and mPACA protocols. Figure 7 defines experiments $\mathsf{Expt}_{\mathsf{PACA}}^{\mathsf{wpriv}}$ and $\mathsf{Expt}_{\mathsf{mPACA}}^{\mathsf{wpriv}}$. The only difference introduced in these weak privacy notions is an additional flag freshDH, which rules out the trivial win where no Reboot is performed to a token involved in any pair of queries to a regular oracle and a LEFT/RIGHT oracle for Bind runs. For protocol prot ∈ {PACA, mPACA}, the weak privacy advantage of $\mathcal{A}$ is defined as:

$$\mathsf{Adv}_{\mathsf{prot}}^{\mathsf{wpriv}}(\mathcal{A}) = |2 \Pr[\mathsf{Expt}_{\mathsf{prot}}^{\mathsf{wpriv}}(\mathcal{A}) = 1] - 1| \ .$$

For CTAP 2.1 and CTAP 2.1+, Public(T) in particular contains the token version, supported PIN/UV Auth Protocol list and PIN retry counter. The following theorem shows that CTAP 2.1 achieves PACA weak privacy and CTAP 2.1+ achieves mPACA weak privacy.

Theorem 3. *For every efficient adversary $\mathcal{A}$ that makes at most $q_S, q_E, q_{Send}, q_{NT}$ and q_R queries to Setup, Execute, Send-Bind-T, NewT and Reboot, and at most q_{Send}^{act} active queries to Send-Bind-T, for some PIN-sampling distribution $\mathcal{D}$ with minimum entropy $h_{\mathcal{D}}$, there exist efficient adversaries $\mathcal{B}_1, \mathcal{B}_4, \mathcal{B}_5, \mathcal{B}_6$ such that:*

$$\begin{aligned} Adv_{prot}^{wpriv}(\mathcal{A}) \leq 2 \cdot [&q_{Send}^{act}/2^{h_{\mathcal{D}}} + (q_S + q_E + q_{Send})\, Adv_{ECDH}^{sCDH}(\mathcal{B}_1) \\ &+ (q_S + q_E + q_{NT} + q_R + 2q_{Send})^2 \ / \ (2q) + Adv_H^{coll}(\mathcal{B}_4) \\ &+ q_S\, Adv_{SKE}^{IND\text{-}1\$PA\text{-}LHPC}(\mathcal{B}_5) + q_E\, Adv_{SKE}^{IND\text{-}1\$PA\text{-}LPC}(\mathcal{B}_6)] \ , \end{aligned}$$

where prot ∈ {CTAP 2.1, CTAP 2.1+}.

Intuitively, this theorem shows that privacy is achieved if the attacker does not observe CTAP traces where a token reuses the same DH share when interacting with clients to register multiple accounts. We give main ideas of the proof here, and delay the full proof to until Theorem 4, as proofs for these two theorems are identical. Interestingly, the proof reuses many of the arguments used to prove the authentication properties of the protocol. This is the case because the access control mechanism of the token, which is crucial to guarantee authentication, also safeguards the token from interactions that might reveal its long-term state (in practice this includes the credentials it stores inside). Therefore, when we follow similar footsteps of authentication proof to switch tokens' PIN to all 0, and pintokens to be random, the two challenged tokens are essentially indistinguishable if we enforce tokens do not reuse DH shares across different phases (freshDH check) and exclude other trivial attacks. We also remark that, as shown in the proof, the additional mPACA token-to-client authentication in CTAP 2.1+

does not affect privacy and hence the privacy security bounds for CTAP 2.1 and CTAP 2.1+ are the same.

5.3 CTAP 2.1++ for Stronger Privacy

We propose a small modification in CTAP 2.1+ so that procedure obtainSharedSecret-T always regenerates and outputs a fresh Diffie-Hellman share. We refer to this modified protocol as CTAP 2.1++, with modification details formally shown in Figure 12. We then prove its privacy guarantees.[7] Here the Public function is the same as that for CTAP 2.1 and CTAP 2.1+. The following theorem states the result. Its proof is in Appendix D.3.

Theorem 4. *For every efficient adversary $\mathcal{A}$ that makes at most q_S, q_E, q_{Send}, q_{NT} and q_R queries to Setup, Execute, Send-Bind-T, NewT and Reboot, and at most q^{act}_{Send} active queries to Send-Bind-T, for some PIN-sampling distribution $\mathcal{D}$ with minimum entropy $h_{\mathcal{D}}$, there exist adversaries $\mathcal{B}_1$, $\mathcal{B}_4$, $\mathcal{B}_5$, $\mathcal{B}_6$ such that:*

$$\begin{aligned} Adv^{priv}_{CTAP\ 2.1++}(\mathcal{A}) &\le 2 \cdot [q^{act}_{Send}/2^{h_{\mathcal{D}}} + (q_S + q_E + q_{Send})\ Adv^{sCDH}_{ECDH}(\mathcal{B}_1) \\ &+ (q_S + q_E + q_{NT} + q_R + 2q_{Send})^2\ /\ (2q) + Adv^{coll}_{H}(\mathcal{B}_4) \\ &+ q_S\ Adv^{IND\text{-}1\$PA\text{-}LHPC}_{SKE}(\mathcal{B}_5) + q_E\ Adv^{IND\text{-}1\$PA\text{-}LPC}_{SKE}(\mathcal{B}_6)]\ . \end{aligned}$$

We remark that the above theorem also applies to CTAP 2.1 once the same modification with respect to token's DH share regeneration is introduced to the protocol.

5.4 Composed Privacy of FIDO2 and of WebAuthn and CTAP 2.1++

PlA privacy model and WebAuthn privacy. In Appendix G.3 we recall the PlA privacy model from [7], discuss how we slightly strengthen and generalize it, and define PlA privacy advantage $\mathsf{Adv}^{\mathsf{priv}}_{\mathsf{PlA}}$. Theorem 7 in Appendix G.3 states the WebAuthn privacy result, for which the proof from [7] still applies. It shows that for any $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{priv}}_{\mathsf{WebAuthn}}(\mathcal{A}) = 0$, for all attestation modes we consider.

Composed privacy model. We define privacy for the composition of PlA and (m)PACA, denoted by PlA+(m)PACA, in order to assess the composed privacy guarantees provided by FIDO2 and the protocol composed with WebAuthn and CTAP 2.1++. The security experiments $\mathsf{Expt}^{\mathsf{com\text{-}priv}}_{\mathsf{PlA+PACA}}$, $\mathsf{Expt}^{\mathsf{com\text{-}priv}}_{\mathsf{PlA+mPACA}}$ are shown in Figure 8.

The composed privacy experiments give the adversary access to all (m)PACA privacy oracles except those Auth and Validate oracles and to all PlA oracles except rResp, aResp, r/aLEFT and r/aRIGHT. Similar to the composed authentication model shown in Appendix H, the above PlA oracles are modified to perform additional (m)PACA token validation before creating the token's PlA response, and perform token authorization before returning the response. Such changes are colored in blue in Figure 8.

At the end of the experiments, the adversary outputs its guess b', and the experiment checks both conditions for the underlying PlA and (m)PACA privacy. In particular, the composed check is the conjunction of Check-priv-PlA (defined in Appendix G.3) and Check-priv-PACA (defined in Figure 7).

[7]Note that CTAP 2.1++ still achieves mPACA authentication security. The only change to the proof given in Appendix D.2 is in bound $|\Pr_2 - \Pr_3|$, which becomes $(2q_S + 2q_E + q_{NT} + q_R + 2q_{Send})^2\ /\ (2q)$. This captures the slight increase in the total number of fresh Diffie-Hellman shares generated in Setup and Execute.

```
Expt^{com-(w)priv}_{PlA+PACA/PlA+mPACA}(𝒜):
  1: $\mathcal{L}^{PlA}_{corr} \leftarrow \emptyset, \mathcal{L}^{r}_{ch} \leftarrow \emptyset, \mathcal{L}^{r}_{lr} \leftarrow \emptyset, \mathcal{L}^{a}_{ch} \leftarrow \emptyset, \mathcal{L}^{a}_{lr} \leftarrow \emptyset, \mathcal{L}^{PACA}_{corr} \leftarrow \emptyset, \mathcal{L}^{bd}_{ch} \leftarrow \emptyset,$
     $\mathcal{L}^{bd}_{lr} \leftarrow \emptyset, \mathcal{L}^{op}_{ch} \leftarrow \emptyset, \mathcal{L}^{op}_{lr} \leftarrow \emptyset$, pb $\leftarrow 1$, freshDH $\leftarrow 1$
  2: st_1 $\xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(1^\lambda)$ // Phase 1
  3: $T_0, T_1, C_0, C_1, U_0, U_1, S_L, S_R$, st_2 $\leftarrow \mathcal{A}(1^\lambda, \mathrm{st}_1)$ // Phase 2
  4: $b \leftarrow$ InitRL($T_0, T_1, C_0, C_1, U_0, U_1, S_L, S_R$)
  5: $\mathcal{O}' \leftarrow \mathcal{O} \setminus \{$NewT, NewToken$\}$
  6: $b' \leftarrow \mathcal{A}^{\mathcal{O}', \mathrm{LEFT}, \mathrm{RIGHT}}(1^\lambda, \mathrm{st}_2)$ // Phase 3
  7: $r \xleftarrow{\$} \{0, 1\}$
  8: if Check-priv-PlA() ∧ Check-(w)priv-PACA() then return $b = b'$
  9: else return $r$

InitRL($T_0, T_1, C_0, C_1, U_0, U_1, S_L, S_R$):
  1: $b \xleftarrow{\$} \{0, 1\}$
  2: Initialize r/aLEFT_{$T_b, S_L$}, r/aRIGHT_{$T_{1-b}, S_R$}
  3: Initialize (m)PACA LEFT/RIGHT oracles
  4: return $b$

rResp' (($T, j$), $j'$, $m$, $t_{cl}$, $d$): // helper function
  1: status $\xleftarrow{\$}$ validate-T($\pi^{j'}_T, m_{rcom}, t_{cl}, d$)
  2: if status ≠ accepted then return ⊥
  3: if $\tilde{\pi}^j_{r,T} \neq \perp$ or $T$.gid $= \perp$ then return ⊥
  4: ($m_{rrsp}$, rc_$T$, cid, sid, agCon) $\xleftarrow{\$}$ rRsp($\tilde{\pi}^j_{r,T}, m_{rcom}$)
  5: ($m_{rrsp}, t_{tk}$) $\xleftarrow{\$}$ auth-T($\pi^{j'}_T, m_{rrsp}$)
  6: $\mathcal{L}^{r}_{lr} \xleftarrow{\cup} \{$cid$\}$, $\mathcal{L}^{op}_{lr} \xleftarrow{\cup} \{(T, j')\}$
  7: return ($m_{rrsp}, t_{tk}$)

aResp' (($T, j$), $j'$, $m$, $t_{cl}$, $d$): // helper function
  1: status $\xleftarrow{\$}$ validate-T($\pi^{j'}_T, m_{acom}, t_{cl}, d$)
  2: if status ≠ accepted then return ⊥
  3: if $\tilde{\pi}^j_{a,T} \neq \perp$ or $T$.gid $= \perp$ then return ⊥
  4: ($m_{arsp}$, rc_$T$, cid, sid, agCon) $\xleftarrow{\$}$ aRsp($\tilde{\pi}^j_{a,T}, m_{acom}$)
  5: ($m_{arsp}, t_{tk}$) $\xleftarrow{\$}$ auth-T($\pi^{j'}_T, m_{arsp}$)
  6: $\mathcal{L}^{a}_{lr} \xleftarrow{\cup} \{$cid$\}$, $\mathcal{L}^{op}_{lr} \xleftarrow{\cup} \{(T, j')\}$
  7: return ($m_{arsp}, t_{tk}$)

rResp ($T, j, j', m_{rcom}, t_{cl}, d$):
  1: status $\xleftarrow{\$}$ validate-T($\pi^{j'}_T, m_{rcom}, t_{cl}, d$)
  2: if status ≠ accepted then return ⊥
  3: if $\tilde{\pi}^j_{r,T} \neq \perp$ or $T$.gid $= \perp$ then return ⊥
  4: ($m_{rrsp}$, rc_$T$, cid, sid, agCon) $\xleftarrow{\$}$ rRsp($\tilde{\pi}^j_{r,T}, m_{rcom}$)
  5: ($m_{rrsp}, t_{tk}$) $\xleftarrow{\$}$ auth-T($\pi^{j'}_T, m_{rrsp}$)
  6: $\mathcal{L}^{r}_{ch} \xleftarrow{\cup} \{$cid$\}$, $\mathcal{L}^{op}_{ch} \xleftarrow{\cup} \{(T, j')\}$
  7: return ($m_{rrsp}, t_{tk}$)

aResp ($T, j, j', m_{acom}, t_{cl}, d$):
  1: status $\xleftarrow{\$}$ validate-T($\pi^{j'}_T, m_{acom}, t_{cl}, d$)
  2: if status ≠ accepted then return ⊥
  3: if $\tilde{\pi}^j_{a,T} \neq \perp$ or $T$.gid $= \perp$ then return ⊥
  4: ($m_{arsp}$, rc_$T$, cid, sid, agCon) $\xleftarrow{\$}$ aRsp($\tilde{\pi}^j_{a,T}, m_{acom}$)
  5: ($m_{arsp}, t_{tk}$) $\xleftarrow{\$}$ auth-T($\pi^{j'}_T, m_{arsp}$)
  6: if tokenBindPartner($T, j'$) ≠ ⊥ then
  7:     ($C, j$) ← tokenBindPartner($T, i$)
  8:     if ($C, j, m_{acom}, t_{cl}$) ∈ $\mathcal{L}_{authC}$ then
  9:         $\mathcal{L}^{a}_{ch} \xleftarrow{\cup} \{$cid$\}$, $\mathcal{L}^{op}_{ch} \xleftarrow{\cup} \{(T, j')\}$
 10: return ($m_{arsp}, t_{tk}$)

r/aLEFT_{$T_b, S_L$}($j'$, $m$, $t_{cl}$, $d$)
  1: Obtains intended server $S$ from $m$
  2: if $S \neq S_L$ then return ⊥
  3: $j \leftarrow 0$, while $\pi^j_{T_b} \neq \perp$: $j \leftarrow j + 1$
  4: return rResp' (($T_b, j$), $j'$, $m$, $t_{cl}$, $d$) // in rLEFT
  5: return aResp' (($T_b, j$), $j'$, $m$, $t_{cl}$, $d$) // in aLEFT

r/aRIGHT_{$T_{1-b}, S_R$}($j'$, $m$, $t_{cl}$, $d$)
  1: Obtains intended server $S$ from $m$
  2: if $S \neq S_R$ then return ⊥
  3: $j \leftarrow 0$, while $\pi^j_{T_{1-b}} \neq \perp$: $j \leftarrow j + 1$
  4: return rResp' (($T_{1-b}, j$), $j'$, $m$, $t_{cl}$, $d$) // in rRIGHT
  5: return aResp' (($T_{1-b}, j$), $j'$, $m$, $t_{cl}$, $d$) // in aRIGHT
```

Figure 8: Composed privacy experiments $\mathsf{Expt}^{\mathsf{com\text{-}priv}}_{\mathsf{PlA+mPACA}}$ (with red code) and $\mathsf{Expt}^{\mathsf{com\text{-}priv}}_{\mathsf{PlA+PACA}}$ (without red code). They each also has a "weak" privacy version, $\mathsf{Expt}^{\mathsf{com\text{-}wpriv}}_{\mathsf{PlA+mPACA}}$ and $\mathsf{Expt}^{\mathsf{com\text{-}wpriv}}_{\mathsf{PlA+PACA}}$, which capture (m)PACA weak privacy. $\mathcal{O}$ contains all (m)PACA privacy oracles (Figure 7) except those Auth and Validate oracles and all PlA oracles (Figure 19 in Appendix G) except rResp, aResp, r/aLEFT and r/aRIGHT. These PlA oracles are re-defined for composed model, where rResp', and aResp' are internal helper functions called inside r/aLEFT and r/aRIGHT; differences are highlighted in blue.

Figure 8 also shows experiments $\mathsf{Expt}^{\mathsf{com\text{-}wpriv}}_{\mathsf{PlA+PACA}}$, $\mathsf{Expt}^{\mathsf{com\text{-}wpriv}}_{\mathsf{PlA+mPACA}}$ that define the weak composed privacy for PlA+(m)PACA protocols to capture (m)PACA weak privacy, where the winning conditions of (m)PACA weak privacy is written as Win-wpriv-PACA.

Advantage measures. For notion ∈ {com-priv, com-wpriv} and prot ∈ {PlA + PACA, PlA + mPACA} we define the advantages as

$$\mathsf{Adv}^{\mathsf{notion}}_{\mathsf{prot}}(\mathcal{A}) = |2 \Pr[\mathsf{Expt}^{\mathsf{com\text{-}priv}}_{\mathsf{prot}}(\mathcal{A}) = 1] - 1|\ .$$

Authentication in composed privacy. As described above, our composed model naturally combines privacy models for PlA and (m)PACA. Here we highlight that it also captures the following privacy attack related to (m)PACA authentication. An adversary can break privacy by first breaking (m)PACA authentication to unlock a token and then ask it to respond to any server authentication requests; this could link the token to its prior registrations. In practice, this means an attacker can steal a token and then try to break into it to correlate its internal state with previously observed FIDO2 runs. As we will show in our theorems, protection against this attack is ensured by (m)PACA authentication security.

Formally, to capture such attacks, we add additional checks inside aResp as shown in Figure 8 also colored in blue (lines 6-9) such that context separation is only enforced when the adversary is interacting with the tokens via an honest client. That is, if the adversary can forge an authorized command to trick the aResp oracle to accept and respond, then it breaks (m)PACA authentication nontrivially and hence the context separation is not enforced.

Composed privacy of WebAuthn and CTAP 2.1++. The following theorem states our composition privacy result for PlA+mPACA, with the proof in Appendix D.4. It shows that composed privacy of PlA+mPACA reduces to the authentication and privacy of (m)PACA and the privacy of PlA.

Theorem 5. *For every efficient $\mathcal{A}$, there exist efficient adversaries $\mathcal{B}_1$, $\mathcal{B}_2$ and $\mathcal{B}_3$ such that:*

$$Adv^{com\text{-}priv}_{PlA+mPACA}(\mathcal{A}) \leq Adv^{SUF\text{-}t}_{mPACA}(\mathcal{B}_1) + Adv^{priv}_{mPACA}(\mathcal{B}_2) + Adv^{priv}_{PlA}(\mathcal{B}_3) \ .$$

The composed privacy of WebAuthn+CTAP 2.1++ follows from the above Theorem 5, together with Theorem 2 adapted to CTAP 2.1++ (CTAP 2.1++ achieves mPACA authentication), Theorem 4 (CTAP 2.1++ achieves mPACA privacy), and Theorem 7 (WebAuthn achieves PlA privacy, shown in Appendix G.3).

Composed privacy of FIDO2. In order to analyze privacy of the original FIDO2, we derive a similar composition theorem for PlA+PACA. with proof (omitted for simplicity) essentially the same as that of Theorem 5, by replacing mPACA results with PACA ones.

Theorem 6. *For every efficient $\mathcal{A}$, there exist efficient adversaries $\mathcal{B}_1$, $\mathcal{B}_2$ and $\mathcal{B}_3$ such that:*

$$Adv^{com\text{-}wpriv}_{PlA+PACA}(\mathcal{A}) \leq Adv^{SUF\text{-}t}_{PACA}(\mathcal{B}_1) + Adv^{wpriv}_{PACA}(\mathcal{B}_2) + Adv^{priv}_{PlA}(\mathcal{B}_3) \ .$$

The composed privacy of the current version of FIDO2 follows from the above Theorem 6, together with Theorems 1 (CTAP 2.1 achieves PACA authentication) , Theorem 3 (CTAP 2.1 achieves PACA weak privacy), and Theorem 7 (WebAuthn achieves PlA privacy, shown in Appendix G.3). The result implies that the unmodified FIDO2 achieves privacy as long as the attacker does not observe CTAP traces where a token reuses the same DH share when interacting with clients to register multiple accounts.

Remark. Again, we note that our results highlight an interesting correlation between privacy and authentication. The results show that the *authentication* properties of CTAP 2.1 play a crucial role in the *privacy* properties of FIDO2 as a whole. Intuitively, CTAP 2.1 enforces an access control mechanism that prevents everyone other than the user who can unlock the token to use the secret signing keys locked inside. If this was not the case, then any process in any machine to which the token is connected could check if a given credential is associated with one of the signing keys stored in the token. To the best of our knowledge, this correlation between privacy and authentication has not been formalized before.

6 Practical considerations

We briefly justify our claim that the fix we propose to CTAP 2.1 has minimal impact in practical implementations, which is why we argue that the security benefits it brings could be easily brought to real-world applications.

We have implemented our modifications to CTAP 2.1 in forks ([11, 12]) of two popular open-source implementations of FIDO2, one for token-side operations and the other for client-side operations. These implementations are respectively maintained by Nitrokey [22] and Mozilla [21]. Our implementations was introduced as a new version of the CTAP protocol, which can be reported as supported by the token and recognized by the client. The new implementation co-exists naturally with previous versions. In terms of changes to the cryptographic cores on both sides of the protocol, we needed to add a new symmetric key expansion step on both sides, plus MAC generation on the token-side and MAC verification on the client-side. However, the impact on the code footprint is very small (0.35% increase in the binary size for the token) because we can reuse pre-existing key expansion and MAC computation code. The bandwidth increase is 32-bytes in token-to-client responses, and the impact on round trip time is 13.4% (the difference is in the range of 0.09 seconds). Note that these results were obtained over a naive adaptation of the code, without any attempt to perform non-trivial optimizations.

Finally, we also investigated the impact of our proposed modification to token-side code to improve privacy, where we require the token to always use a fresh DH share. We implemented the most naive solution, which is to generate the new DH share just before it is transmitted to the client, and measured the impact on round-trip time. We measured the overhead at around 0.6% (less than 0.01 seconds), which is imperceptible to human users.

7 Conclusion

We revisit the privacy and security of FIDO2 by focusing on the role of the CTAP component. We look for the first time at the impact of CTAP on privacy, and we clarify the contribution of CTAP on authentication properties. We show that, by improving CTAP, one can mitigate rogue key attacks and other related attacks. For this, we propose a simple fix that has very limited impact on code base and performance: adding a symmetric key expansion step to obtain two MAC keys instead of one, and an extra MAC computation to protect messages sent from the token to the client. On the privacy front, we show that the only thing that can compromise security is the reuse of DH shares, and this can be prevented with a simple CTAP protocol change or by the user ensuring that the token is rebooted between registrations.

Acknowledgments

Shan Chen was supported by the research start-up grant from the Southern University of Science and Technology. Alexandra Boldyreva and Kaishuo Cheng were supported by the National Science Foundation under Grant No.1946919. Luís Esquível was supported by Fundação para a Ciência e Tecnologia (FCT), Portugal - 2021.07225.BD.

References

[1] FIDO Alliance. 2022. Client to Authenticator Protocol (CTAP) – Proposed Standard. https://fidoalliance.org/specs/fido-v2.1-ps-20210615/fido-client-to-authenticator-protocol-v2.1-ps-errata-20220621.html.
[2] Manuel Barbosa, Alexandra Boldyreva, Shan Chen, Kaishuo Cheng, and Luís Esquível. 2025. Privacy and Security of FIDO2 Revisited. Cryptology ePrint Archive, Paper 2025/459. https://eprint.iacr.org/2025/459
[3] Manuel Barbosa, Alexandra Boldyreva, Shan Chen, and Bogdan Warinschi. 2021. Provable Security Analysis of FIDO2. In *Advances in Cryptology - CRYPTO 2021 (Lecture Notes in Computer Science, Vol. 12827)*. Springer, 125–156. https://doi.org/10.1007/978-3-030-84252-9_5 Full version: https://eprint.iacr.org/2020/756.
[4] Manuel Barbosa, André Cirne, and Luís Esquível. 2023. Rogue key and impersonation attacks on FIDO2: From theory to practice. In *Proceedings of the 18th International Conference on Availability, Reliability and Security, ARES 2023, Benevento, Italy, 29 August 2023- 1 September 2023*. ACM, 14:1–14:11. https://doi.org/10.1145/3600160.3600174
[5] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. 1996. Keying hash functions for message authentication. In *CRYPTO 1996*. Springer, 1–15.
[6] Nina Bindel, Cas Cremers, and Mang Zhao. 2023. FIDO2, CTAP 2.1, and WebAuthn 2: Provable security and post-quantum instantiation. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1471–1490.
[7] Nina Bindel, Nicolas Gama, Sandra Guasch, and Eyal Ronen. 2023. To attest or not to attest, this is the question–Provable attestation in FIDO2. In *ASIACRYPT 2023*. Springer, 297–328.
[8] Brice Canvel, Alain P. Hiltgen, Serge Vaudenay, and Martin Vuagnoux. 2003. Password Interception in a SSL/TLS Channel. In *Advances in Cryptology - CRYPTO 2003 (Lecture Notes in Computer Science, Vol. 2729)*. Springer, 583–599. https://doi.org/10.1007/978-3-540-45146-4_34
[9] Robert Dumitru, Daniel Genkin, Andrew Wabnitz, and Yuval Yarom. 2023. The Impostor Among US(B): Off-Path Injection Attacks on USB Communications. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, Anaheim, CA, 5863–5880. https://www.usenix.org/conference/usenixsecurity23/presentation/dumitru
[10] M Dworkin. 2001. Recommendation for Block Cipher Modes of Operation. *Methods and Techniques* (2001).
[11] Luís Esquível. 2025. authenticator-rs-fork. https://github.com/esquivel71/authenticator-rs_fork.
[12] Luís Esquível. 2025. nitrokey-3-firmware-fork. https://github.com/esquivel71/nitrokey-3-firmware_fork.
[13] FIDO Alliance. 2019. Client to Authenticator Protocol (CTAP) – Proposed Standard. https://fidoalliance.org/specs/fido-v2.0-ps-20190130/fido-client-to-authenticator-protocol-v2.0-ps-20190130.html.
[14] FIDO Alliance. Accessed August 2024. User Authentication Specifications Overview. https://fidoalliance.org/specifications/.
[15] Marc Fischlin and Arno Mittelbach. 2021. An Overview of the Hybrid Argument. Cryptology ePrint Archive, Report 2021/088. https://eprint.iacr.org/2021/088.
[16] Iness Ben Guirat and Harry Halpin. 2018. Formal verification of the W3C web authentication protocol. In *5th Annual Symposium and Bootcamp on Hot Topics in the Science of Security*. ACM, 6.
[17] Lucjan Hanzlik, Julian Loss, and Benedikt Wagner. 2023. Token meets wallet: Formalizing privacy and revocation for FIDO2. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1491–1508.
[18] Kevin Igoe, David McGrew, and Margaret Salter. 2011. Fundamental Elliptic Curve Cryptography Algorithms. RFC 6090. https://doi.org/10.17487/RFC6090
[19] Michal Kepkowski, Lucjan Hanzlik, Ian D Wood, and Mohamed Ali Kaafar. 2022. How Not to Handle Keys: Timing Attacks on FIDO Authenticator Privacy. *Proceedings on Privacy Enhancing Technologies* (2022).
[20] Dr. Hugo Krawczyk and Pasi Eronen. 2010. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869. https://doi.org/10.17487/RFC5869
[21] Mozilla. 2025. authenticator-rs. https://github.com/mozilla/authenticator-rs.
[22] Nitrokey. 2025. nitrokey-3-firmware. https://github.com/Nitrokey/nitrokey-3-firmware.
[23] Victor Shoup. 2004. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Paper 2004/332. https://eprint.iacr.org/2004/332
[24] W3C. 2021. Web Authentication: An API for accessing Public Key Credentials Level 2 – W3C Recommendation. https://www.w3.org/TR/2021/REC-webauthn-2-20210408/.
[25] W3C. 2023. Web Authentication: An API for accessing Public Key Credentials Level 3 – W3C Recommendation. https://www.w3.org/TR/webauthn-3/.

A Description of WebAuthn

This section describes the protocol algorithms of WebAuthn, as shown in Figure 9, which follows the descriptions presented in [6, 7]. For better presentation, we omit some details that are irrelevant to the attestation modes considered in this work: None, Self, Basic.

Recall that, as shown in Figure 2, WebAuthn has two challenge-response flows, one for registering a new credential with the server, with algorithms rChal, rCom, rRsp, and rVrfy, and one for authenticating under a previously registered credential, with algorithms aChal, aCom, aRsp, and aVrfy.

On registration, the server runs rChal, which inputs the server identity id_S, token binding state tb and user verification condition UV (indicatign whether the user should be verified with PIN or biometrics), and samples a new random challenge ch_S and user identifier uid_S. All these variables, except the token binding state, compose the challenge message m_{rch}, which is output and delivered to the client C. C then runs rCom, which inputs the server domain $\hat{\mathsf{id}}_s$, the token binding state tb and the challenge message m_{rch}, and first checks if the received server id_S identity in m_{rch} matches $\hat{\mathsf{id}}_s$. If it does, then it combines the server challenge ch and the token binding state tb into a client message m_{rcl}, which is then hashed. The digest h, along with the server identity id_S, the user identifier uid and the user verification condition UV, are grouped into a command message m_{rcom}, which is output and sent to the authenticator T. The authenticator runs rRsp, which inputs m_{rcom} and generates a new assertion key pair (pk, sk), samples a new credential identifier cid and sets the signature counter n to zero. Next, it combines the received id_S, the signature counter n, the cid, the new public key pk and the received UV into a message m and, depending on the attestation mode required by relying party (Basic in Figure. 2), produces a new attestation signature σ_{att} on m and h (the hash of m_{rcl}) using the authenticator's private attestation ak. The authenticator then sets, in its registration context rc_T, and for the server identity id_S, the received user identifier uid, the sampled cid, the assertion secret key sk and the signature counter n, and also sets the id_S, h, cid, n, pk, UV and attestation mode used as the agreed content agCon (data that must be the same both from the perspective of the server and the token). Finally, it sets the hash of id_S, the cid and the signature counter as the session identifier sid, and returns the token response message m_{rrsp}, composed by m and (optionally) σ_{att} back to the client, which in turn forwards m_{rrsp} and its own m_{rcl} to the server. To conclude the registration, the server runs rVrfy, which inputs id_S, m_{rcl}, m_{rrsp} and public parameters gpars, and then checks if the information that came from the token and client is correct, including the server identity id_S, the token binding state tb_S, the sampled challenge ch_S, the user verification condition UV_S, the signature counter n (which must be zero) and, depending on the attestation mode, the attestation signature σ_{att}. If everything is correct, it sets the agreed content agCon and the session identifier id_S identically to the token, sets the user identifier uid_S, the token assertion public key pk and the public signature counter n in its registration context rc_S, for the specified cid, and finishes by returns successfully.

On authentication, the procedure is very similar. The server starts with aChal, which is identical to rChal except that it does not sample a new uid_S, which is therefore excluded from the output server message m_{ach}. The same is true of the aCom algorithm, which differs only in the output message m_{acom} that no longer contains uid. The token, upon receiving m_{acom}, runs aRsp, which

```
$\mathsf{rChal}(\mathsf{id}_S, \mathsf{tb}, \mathsf{UV})$:
  1: $\mathsf{ch}_S \xleftarrow{\$} \{0,1\}^{\geq\lambda}, \mathsf{tb}_S \leftarrow \mathsf{tb}$
  2: $\mathsf{UV}_S \leftarrow \mathsf{UV}, \mathsf{uid}_S \xleftarrow{\$} \{0,1\}^{\leq 4\lambda}$
  3: $m_{\mathsf{rch}} \leftarrow (\mathsf{id}_S, \mathsf{ch}_S, \mathsf{uid}_S, \mathsf{UV}_S)$
  4: return $m_{\mathsf{rch}}$
$\mathsf{rCom}(\hat{\mathsf{id}}_S, \mathsf{tb}, m_{\mathsf{rch}})$:
  1: $(\mathsf{id}_S, \mathsf{ch}, \mathsf{uid}, \mathsf{UV}) \leftarrow m_{\mathsf{rch}}$
  2: if $\mathsf{id}_S \neq \hat{\mathsf{id}}_S$ then
  3:     abort
  4: $m_{\mathsf{rcl}} \leftarrow (\mathsf{ch}, \mathsf{tb})$
  5: $h \leftarrow \mathsf{H}(m_{\mathsf{rcl}})$
  6: $m_{\mathsf{rcom}} \leftarrow (\mathsf{id}_S, \mathsf{uid}, h, \mathsf{UV})$
  7: return $(m_{\mathsf{rcl}}, m_{\mathsf{rcom}})$
$\mathsf{rRsp}(T, m_{\mathsf{rcom}})$:
  1: $(\mathsf{id}_S, \mathsf{uid}, h, \mathsf{UV}) \leftarrow m_{\mathsf{rcom}}$
  2: $(pk, sk) \xleftarrow{\$} \mathsf{KG}(), \mathsf{cid} \xleftarrow{\$} \{0,1\}^{\geq\lambda}, n \leftarrow 0$
  3: $m \leftarrow (\mathsf{H}(\mathsf{id}_S), n, \mathsf{cid}, pk, \mathsf{UV})$
  4: $\sigma_{\mathsf{att}} \xleftarrow{\$} \mathsf{Sign}(\mathsf{rc}_T.\mathsf{ak}, (m, h))$
  5: $m_{\mathsf{rrsp}} \leftarrow (m, \sigma_{\mathsf{att}})$
  6: $\mathsf{rc}_T[\mathsf{id}_S] \leftarrow (\mathsf{uid}, \mathsf{cid}, sk, n)$
  7: $\mathsf{agCon} \leftarrow (\mathsf{id}_S, h, \mathsf{cid}, n, pk, \mathsf{UV}, \mathsf{basic})$
  8: $\mathsf{sid} \leftarrow (\mathsf{H}(\mathsf{id}_S), \mathsf{cid}, n)$
  9: return $(m_{\mathsf{rrsp}}, \mathsf{rc}_T, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon})$
$\mathsf{rVrfy}(\mathsf{id}_S, m_{\mathsf{rcl}}, m_{\mathsf{rrsp}}, \mathsf{gpars})$:
  1: $(\mathsf{ch}, \mathsf{tb}) \leftarrow m_{\mathsf{rcl}}, (m, \sigma_{\mathsf{att}}) \leftarrow m_{\mathsf{rrsp}}$
  2: $(h, n, \mathsf{cid}, pk, \mathsf{UV}) \leftarrow m$
  3: if $\mathsf{ch}_S \neq \mathsf{ch} \vee \mathsf{tb}_S \neq \mathsf{tb} \vee h \neq \mathsf{H}(\mathsf{id}_S) \vee \mathsf{UV}_S \neq \mathsf{UV} \vee n \neq 0 \vee \mathsf{Ver}(\mathsf{gpars.vk}, (m, \mathsf{H}(\mathsf{id}_S)), \sigma_{\mathsf{att}}) = 0$ then
  4:     return $(0, \mathsf{rc}_S, \perp, \perp, \perp)$
  5: $\mathsf{agCon} \leftarrow (\mathsf{id}_S, \mathsf{H}(m_{\mathsf{rcl}}), \mathsf{cid}, n, pk, \mathsf{UV}, \mathsf{basic})$
  6: $\mathsf{rc}_S[\mathsf{cid}] \leftarrow (\mathsf{uid}_S, pk, n)$
  7: $\mathsf{sid} \leftarrow (\mathsf{H}(\mathsf{id}_S), \mathsf{cid}, n)$
  8: return $(1, \mathsf{rc}_S, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon})$
```

```
$\mathsf{aChal}(\mathsf{id}_S, \mathsf{tb}, \mathsf{UV})$:
  1: $\mathsf{ch}_S \xleftarrow{\$} \{0,1\}^{\geq\lambda}, \mathsf{tb}_S \leftarrow \mathsf{tb}$
  2: $\mathsf{UV}_S \leftarrow \mathsf{UV}$
  3: $m_{\mathsf{ach}} \leftarrow (\mathsf{id}_S, \mathsf{ch}_S, \mathsf{UV}_S)$
  4: return $m_{\mathsf{ach}}$
$\mathsf{aCom}(\bar{\mathsf{id}}_S, \mathsf{tb}, m_{\mathsf{ach}})$:
  1: $(\mathsf{id}_S, \mathsf{ch}, \mathsf{UV}) \leftarrow m_{\mathsf{ach}}$
  2: if $\mathsf{id}_S \neq \bar{\mathsf{id}}_S$ then
  3:     abort
  4: $m_{\mathsf{acl}} \leftarrow (\mathsf{ch}, \mathsf{tb})$
  5: $h \leftarrow \mathsf{H}(m_{\mathsf{acl}})$
  6: $m_{\mathsf{acom}} \leftarrow (\mathsf{id}_S, h, \mathsf{UV})$
  7: return $(m_{\mathsf{acl}}, m_{\mathsf{acom}})$
$\mathsf{aRsp}(T, m_{\mathsf{acom}})$:
  1: $(\mathsf{id}_S, h, \mathsf{UV}) \leftarrow m_{\mathsf{acom}}$
  2: $\mathsf{rc}_T[\mathsf{id}_S].n \leftarrow \mathsf{rc}_T[\mathsf{id}_S].n + 1$
  3: $ad \leftarrow (\mathsf{H}(\mathsf{id}_S), \mathsf{rc}_T[\mathsf{id}_S].n, \mathsf{UV})$
  4: $\sigma \leftarrow \mathsf{Sign}(\mathsf{rc}_T[\mathsf{id}_S].sk, (ad, h))$
  5: $m_{\mathsf{arsp}} \leftarrow (\mathsf{rc}_T[\mathsf{id}_S].\mathsf{cid}, ad, \sigma, \mathsf{rc}_T[\mathsf{id}_S].\mathsf{uid})$
  6: $\mathsf{agCon} \leftarrow (\mathsf{id}_S, h, \mathsf{rc}_T[\mathsf{id}_S].n, \mathsf{UV})$
  7: $\mathsf{sid} \leftarrow (\mathsf{H}(\mathsf{id}_S), \mathsf{rc}_T[\mathsf{id}_S].\mathsf{cid}, h, \mathsf{rc}_T[\mathsf{id}_S].n)$
  8: return $(m_{\mathsf{arsp}}, \mathsf{rc}_T, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon})$

$\mathsf{aVrfy}(\mathsf{id}_S, m_{\mathsf{acl}}, m_{\mathsf{arsp}})$:
  1: $(\mathsf{ch}, \mathsf{tb}) \leftarrow m_{\mathsf{acl}}, (\mathsf{cid}, ad, \sigma, \mathsf{uid}) \leftarrow m_{\mathsf{arsp}}$
  2: $(h, n, \mathsf{UV}) \leftarrow ad$
  3: if $\mathsf{ch}_S \neq \mathsf{ch} \vee \mathsf{tb}_S \neq \mathsf{tb} \vee h \neq \mathsf{H}(\mathsf{id}_S) \vee \mathsf{UV}_S \neq \mathsf{UV} \vee n \leq \mathsf{rc}_S[\mathsf{cid}].n \vee \mathsf{Ver}(\mathsf{rc}_S[\mathsf{cid}].pk, (ad, \mathsf{H}(m_{\mathsf{acl}})), \sigma) = 0$ then
  4:     return $(0, \mathsf{rc}_S, \perp, \perp, \perp)$
  5: $\mathsf{agCon} \leftarrow (\mathsf{id}_S, \mathsf{H}(m_{\mathsf{acl}}), n, \mathsf{UV})$
  6: $\mathsf{rc}_S[\mathsf{cid}].n \leftarrow n$
  7: $\mathsf{sid} \leftarrow (h, \mathsf{cid}, \mathsf{H}(m_{\mathsf{acl}}), n)$
  8: return $(1, \mathsf{rc}_S, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon})$
```

Figure 9: WebAuthn protocol functions.

first increments the signature counter n stored in its registration context. Next, the hashed server identity id_S, along with n, the user verification condition UV and the hashed m_{acl} client message h are signed using the assertion secret key sk generated in the previous registration run to produce a signature σ. Then, it sets the id_S, h, n and UV as the agreed content agCon and the hashed id_S, the cid (sampled and stored in the previous registration run), h and n as the session identifier sid. Finally, it sets the cid, the hashed id_S, n, UV, the assertion signature σ and the user identifier uid as the response message m_{arsp}, which is returned to the client, and then forwarded, along with the client message m_{acl}, to the server. The aVrfy algorithm is also similar to rVrfy. It inputs m_{acl} and m_{arsp} and verifies if the information received is correct, with the major difference being that it now verifies the assertion signature σ using the previously stored assertion public key pk, and also checks if the signature counter received is not greater than the counter stored in the previous registration/authentication session. If everything is correct, it sets the agCon and sid identically to the token, updates the signature counter n, and returns successfully.

B Descriptions of CTAP 2.1, CTAP 2.1+, CTAP 2.1++

High-level flow of CTAP 2.1. We refer here to Figure 3 for the high-level flow of the protocol, which proceeds as follows.

Reboot is performed via the authPowerUp-T function, which inputs the state of the token st_T and freshly samples a new ECDH key pair and pinToken for each supported protocol. It also resets the consecutive pin attempts counter $\mathsf{st}_T.m$. The variable st_T.initialData contains the token version and supported PIN/UV Auth Protocol list, and is used only the first time authPowerUp-T is called (for a token T) to set st_T.version and st_T.puvProtocolList.

During the Setup phase, the authenticator T first outputs its info, which contains the list of supported PIN/UV Auth Protocol (max. 2) and sends it to the client C. C runs obtainSharedSecret-C-start, which selects the protocol it is going to use and outputs it to T. T then runs obtainSharedSecret-T, sets its puvProtocol chosen by C and outputs its ECDH share pk_T back to C. At this stage, C runs two different functions. First, it executes obtainSharedSecret-C-end, which runs the puvProtocol function encapsulate. This function derives the shared secret K from the client's ECDH share and the received share pk_T from the token, and outputs c, which is the client's ECDH share. Then, C runs setPIN-C, which inputs the user pin, encrypts it with K to create ciphertext c_p, and then authenticates c_p to create t_p. In the end, C sends c, c_p, t_p to the token. Finally, after receiving this data, the token executes setPIN-T, which runs puvProtocol function decapsulate to derive the same shared secret K, and uses it to verify and decrypt c_p. The resulting pin is then hashed and stored in the token's static storage st_T. The token's retry counter st_T.pinRetries is also initialized to pinRetriesMax (which is at most 8).

The Bind phase is essentially identical to Setup until the execution of obtainSharedSecret-C-end on the client's side. Afterwards, C runs obtainPinUvAuthToken-C-start, which encrypts the hash of the user pin, and sends its ECDH share and the resulting ciphertext c and c_{ph} to token T. The token executes obtainPinUvAuthToken-T, which, as long as st_T.pinRetries is not 0, runs decapsulate to obtain K, decrements st_T.pinRetries, decrypts c_{ph} and then verifies if the result matches the saved pinHash from Setup. If it does not, then it regenerates the token's ECDH share for the currently in use puvProtocol, decrements the token's consecutive tries counter $\mathsf{st}_T.m$ and, if $\mathsf{st}_T.m$ reaches 0, forces a token reboot. If verification succeeds, then it samples a new pinToken for every supported PIN/UV Auth Protocol, sets the correct pinToken as its binding state, encrypts it with K and sends the resulting ciphertext c_{pt} to the client. Finally, C runs obtainSharedSecret-C-end, which decrypts c_{pt} and sets the result as its binding state.

After a client C and token T have finished Bind, C can use its binding state π_C^j.bs as the key to authenticate a command M by running auth, which outputs M and a tag t, and then the token can validate (M, t) with the same binding state π_T^i.bs by running validate.

We present the full code-based description of all CTAP 2.1 functions in Figure 12.

Session and protocol variables. As in [6], we specify here the relevant variables for tokens and clients specific for CTAP 2.1. All variables defined for PACA are inherited in CTAP 2.1. A token's internal state st_T is composed of: (i) a token version st_T.version (e.g., CTAP2.0 or CTAP2.1), (ii) a list of available PIN/UV protocols st_T.puvProtocolList, (iii) the currently selected PIN/UV protocol st_T.puvProtocol, (iv) the counter for the maximum amount of pin failed attempts st_T.pinRetries, (v) the counter for the number of consecutive pin failed attempts $\mathsf{st}_T.m$ and (vi) the stored hashed pin st_T.pinHash. Both tokens and clients (i.e. all session oracles π_C^j and π_T^i) share: (i) the binding state bs which is set as the pinToken and (ii) the session identifier sid, which is defined as the full trace

```
initialize ():
  1: regenerate()
  2: resetPuvToken()

expand ($pt$):
  1: $pt_e \leftarrow \mathcal{H}_3(pt)$
  2: Parse$(K_C, K_T) \leftarrow pt_e$, s.t. $|K_C| = 2\lambda$
  3: return $(K_C, K_T)$

encrypt $(K, m)$:
  1: Parse$(K_1, K_2) \leftarrow K$, s.t. $|K_1| = 2\lambda$
  2: $c \leftarrow$ SKE.Enc$(K_2, m)$
  3: return $c$

encapsulate $(pk')$:
  1: $Z \leftarrow$ XCoordinateOf$(sk \cdot pk')$
  2: $K_1 \leftarrow \mathcal{H}_2(Z, $"CTAP2 HMAC KEY"$)$
  3: $K_2 \leftarrow \mathcal{H}_2(Z, $"CTAP2 AES KEY"$)$
  4: $K \leftarrow (K_1, K_2)$
  5: $c \leftarrow pk$
  6: return $(c, K)$

authenticate $(K', m)$:
  1: Parse$(K'_1, K'_2) \leftarrow K'$, s.t. $|K'_1| = 2\lambda$
  2: $t \leftarrow$ MAC$(K'_1, m)$
  3: return $t$

getPublicKey ():
  1: return $pk$

regenerate ():
  1: $(pk, sk) \xleftarrow{\$}$ ECDH.KG()

resetPuvToken ():
  1: $pt \xleftarrow{\$} \{0,1\}^{2\lambda}$

decrypt $(K, c)$:
  1: Parse$(K_1, K_2) \leftarrow K$, s.t. $|K_1| = 2\lambda$
  2: $m \leftarrow$ SKE.Dec$(K_2, c)$
  3: return $m$

decapsulate $(c)$:
  1: $Z \leftarrow$ XCoordinateOf$(sk \cdot c)$
  2: $K_1 \leftarrow \mathcal{H}_2(Z, $"CTAP2 HMAC KEY"$)$
  3: $K_2 \leftarrow \mathcal{H}_2(Z, $"CTAP2 AES KEY"$)$
  4: $K \leftarrow (K_1, K_2)$
  5: return $K$

verify $(K', m, t)$:
  1: Parse$(K'_1, K'_2) \leftarrow K'$, s.t. $|K'_1| = 2\lambda$
  2: $t \leftarrow$ MAC$(K'_1, m)$
  3: return $[[t = t']]$
```

Figure 10: Pin UV Auth Protocol 2. All original functions are presented as shown in [6], with our proposed addition of expand for CTAP 2.1+ in blue.

```
encrypt $(K, m)$:
  1: $c \leftarrow$ SKE.Enc$(K, m)$
  2: return $c$

encapsulate $(pk')$:
  1: $Z \leftarrow$ XCoordinateOf$(sk \cdot pk')$
  2: $K \leftarrow \mathcal{H}_1(Z)$
  3: $c \leftarrow pk$
  4: return $(c, K)$

authenticate $(K', m)$:
  1: $t \leftarrow$ MAC$(K', m)$
  2: return $t$

decrypt $(K, c)$:
  1: $m \leftarrow$ SKE.Dec$(K, c)$
  2: return $m$

resetPuvToken ():
  1: $pt \xleftarrow{\$} \{0,1\}^{\mu\lambda}$

decapsulate $(c)$:
  1: $Z \leftarrow$ XCoordinateOf$(sk \cdot c)$
  2: $K \leftarrow \mathcal{H}_1(Z)$
  3: return $K$

verify $(K', m, t)$:
  1: $t \leftarrow$ MAC$(K', m)$
  2: return $[[t = t']]$
```

Figure 11: PIN/UV Auth Protocol 1. Only functions that are different from PIN/UV Auth Protocol 2 are shown.

of Bind. Client sessions have: (i) the selected PIN/UV protocol π^j_C.puvProtocol, and (ii) the ephemeral session key derived from ECDH π^j_C.K. Additionally, we added a new π^i_T.canValidate variable to token sessions, which we explain later in this section.

Low-level CTAP 2.1 description. The original CTAP 2.1 protocol specifies an abstract PIN/UV Auth Protocol, which provides an interface for a set of lower level cryptographic functions, and provides two distinct instantiations, referred to as PIN/UV Auth Protocol 1 and PIN/UV Auth Protocol 2 (seen in Figures 11 and 10 respectively).[8] We start by first describing PIN/UV Auth Protocol 2, which is the protocol we considered for our CTAP 2.1 security analysis and results, and thus adapted for CTAP 2.1+, and then describe PIN/UV Auth Protocol 1 for completeness.

The initialize function, which is called by a token on reboot or a client when starting a new Bind run, generates a fresh ECDH key pair over the NIST P-256 curve and a fresh 2λ-bit pinToken (with $\lambda = 128$). Both actions can also be executed separately via regenerate and resetPuvToken, respectively. The generated ECDH share can then be obtained (but not regenerated) via getPublicKey.

[8]For CTAP 2.1+, we extend only PIN/UV Auth Protocol 2 (which we simply call PIN/UV Auth Protocol) by adding a new function expand.

```
authPowerUp-T(st$_T$):
  1: if st$_T$.version $=\perp \wedge$ st$_T$.puvProtocolList $=\perp$ then
  2:     (version, puvProtocolList) $\leftarrow$ st$_T$.initialData
  3:     st$_T$.version $\leftarrow$ version
  4:     st$_T$.puvProtocolList $\leftarrow$ puvProtocolList
  5: for all puvProtocol $\in$ st$_T$.puvProtocolList do
  6:     st$_T$.puvProtocol.initialize()
  7: st$_T.m \leftarrow 3$

obtainSharedSecret-C-start($\pi^j_C$, info):
  1: Parse(version, puvProtocolList) $\leftarrow$ info
  2: if version = 2.0 then return $\perp$
  3: select puvProtocol $\leftarrow$ puvProtocolList
  4: $\pi^j_C$.puvProtocol $\leftarrow$ puvProtocol
  5: $\pi^j_C$.puvProtocol.initialize()
  6: $\pi^j_C$.st$_{\text{exe}}$ $\leftarrow$ waiting
  7: $\pi^j_C$.sid $\leftarrow \pi^j_C$.sid || info || puvProtocol
  8: return puvProtocol

obtainPinUvAuthToken-C-start($\pi^j_C$, pin):
  1: pinHash $\leftarrow$ H(pin)
  2: $c_{ph} \xleftarrow{\$} \pi^j_C$.puvProtocol.encrypt($\pi^j_C$.K, pinHash)
  3: $\pi^j_C$.st$_{\text{exe}}$ $\leftarrow$ bindStart
  4: $\pi^j_C$.sid $\leftarrow \pi^j_C$.sid || $c_{ph}$
  5: return $c_{ph}$

obtainPinUvAuthToken-C-end($\pi^j_C$, $c_{pt}$):
  1: $\pi^j_C$.bs $\leftarrow \pi^j_C$.puvProtocol.decrypt($\pi^j_C$.K, $c_{pt}$)
  2: $\pi^j_C$.st$_{\text{exe}}$ $\leftarrow$ bindDone
  3: $\pi^j_C$.sid $\leftarrow \pi^j_C$.sid || $c_{pt}$ || false
  4: return

setPIN-C($\pi^j_C$, pin$_U$):
  1: if pin$_U \notin \mathcal{P}$ then return $\perp$
  2: $c_p \xleftarrow{\$} \pi^j_C$.puvProtocol.encrypt($\pi^j_C$.K, pin$_U$)
  3: $t_p \xleftarrow{\$} \pi^j_C$.puvProtocol.authenticate($\pi^j_C$.K, $c_p$)
  4: return $(c_p, t_p)$

setPIN-T($\pi^i_T$, puvProtocol, $c$, $c_p$, $t_p$):
  1: if puvProtocol $\notin$ st$_T$.puvProtocolList $\vee$
     st$_T$.pinHash $\neq \perp$ then
  2:     return $\perp$
  3: $K \leftarrow$ st$_T$.puvProtocol.decapsulate($c$)
  4: if $K = \perp \vee$ st$_T$.puvProtocol.verify($K, c_p, t_p$) =
     false then
  5:     return $\perp$
  6: pin $\leftarrow$ st$_T$.puvProtocol.decrypt($K, c_p$)
  7: if pin $\notin \mathcal{P}$ then return $\perp$
  8: st$_T$.pinHash $\leftarrow$ H(pin)
  9: st$_T$.pinRetries $\leftarrow$ pinRetriesMax
 10: return accepted

auth-C ($\pi^j_C, M$):
  1: $t \xleftarrow{\$} \pi^j_C$.puvProtocol.authenticate($\pi^j_C$.bs, $M$)
  2: return $(M, t)$

— CTAP 2.1+ below
auth-C ($\pi^j_C, M$):
  1: $(K_{\text{authC}}, \_) \leftarrow \pi^j_C$.puvProtocol.expand($\pi^j_C$.bs)
  2: $t \xleftarrow{\$} \pi^j_C$.puvProtocol.authenticate($K_{\text{authC}}, M$)
  3: return $(M, t)$

validate-C ($\pi^j_C, M, t$):
  1: $(\_, K_{\text{authT}}) \leftarrow \pi^j_C$.puvProtocol.expand($\pi^j_C$.bs)
  2: if $\pi^j_C$.puvProtocol.verify($K_{\text{authT}}, M, t$) = true
     then return accepted
  3: return rejected

getInfo-T($\pi^i_T$):
  1: info $\leftarrow$ (st$_T$.version, st$_T$.puvProtocolList)
  2: $\pi^i_T$.sid $\leftarrow \pi^i_T$.sid || info

obtainSharedSecret-T($\pi^i_T$, puvProtocol):
  1: if puvProtocol $\notin$ st$_T$.puvProtocolList then return
     $\perp$
  2: st$_T$.puvProtocol.regenerate()
  3: $pk_T \leftarrow$ st$_T$.puvProtocol.getPublicKey()
  4: $\pi^i_T$.st$_{\text{exe}}$ $\leftarrow$ waiting
  5: $\pi^i_T$.sid $\leftarrow \pi^i_T$.sid || puvProtocol || $pk_T$
  6: return $pk_T$

obtainSharedSecret-C-end($\pi^j_C$, $pk_T$):
  1: $(c, K) \leftarrow \pi^j_C$.puvProtocol.encapsulate($pk_T$)
  2: $\pi^j_C$.K $\leftarrow K$
  3: $\pi^j_C$.sid $\leftarrow \pi^j_C$.sid || $pk_T$ || puvProtocol || $c$
  4: return $c$

obtainPinUvAuthToken-T($\pi^i_T$, puvProtocol, $c$, $c_{ph}$):
  1: if puvProtocol $\notin$ st$_T$.puvProtocolList $\vee$
     st$_T$.pinRetries = 0 then
  2:     return ($\perp$, false)
  3: $K \leftarrow$ st$_T$.puvProtocol.decapsulate($c$)
  4: if $K = \perp$ then return ($\perp$, false)
  5: st$_T$.pinRetries $\leftarrow$ pinRetries $-$ 1
  6: pinHash $\leftarrow$ st$_T$.puvProtocol.decrypt($K, c_{ph}$)
  7: if pinHash $\neq$ st$_T$.pinHash then
  8:     st$_T$.puvProtocol.regenerate()
  9:     st$_T.m \leftarrow$ st$_T.m - 1$
 10:     if st$_T.m = 0$ then
 11:         authPowerUp-T(st$_T$)
 12:         return ($\perp$, true)
 13:     else
 14:         return ($\perp$, false)
 15: st$_T.m \leftarrow 3$
 16: st$_T$.pinRetries $\leftarrow$ pinRetriesMax
 17: for all puvProtocol' $\in$ st$_T$.puvProtocolList do
 18:     st$_T$.puvProtocol'.resetPuvToken()
 19: $\pi^i_T$.bs $\leftarrow \pi^i_T$.puvProtocol.$pt$
 20: $c_{pt} \xleftarrow{\$}$ st$_T$.puvProtocol.encrypt($K, \pi^i_T$.bs)
 21: $\pi^i_T$.st$_{\text{exe}}$ $\leftarrow$ bindDone
 22: $\pi^i_T$.canValidate $\leftarrow$ true
 23: $\pi^i_T$.sid $\leftarrow \pi^i_T$.sid || puvProtocol || $c$ || $c_{ph}$ || $c_{pt}$ ||
     false
 24: return ($c_{pt}$, false)

validate-T ($\pi^i_T, M, t, d$):
  1: if st$_T$.puvProtocol.verify($\pi^i_T$.bs, $M, t$) = true then
  2:     return $d$
  3: return rejected

— CTAP 2.1+ below
auth-T ($\pi^i_T, M$):
  1: $(\_, K_{\text{authT}}) \leftarrow \pi^j_C$.puvProtocol.expand($\pi^i_T$.bs)
  2: $t \xleftarrow{\$}$ st$_T$.puvProtocol.authenticate($K_{\text{authT}}, M$)
  3: return $(M, t)$

validate-T ($\pi^i_T, M, t, d$):
  1: $(K_{\text{authC}}, \_) \leftarrow \pi^j_C$.puvProtocol.expand($\pi^i_T$.bs)
  2: if $\pi^i_T$.canValidate = true then
  3:     if st$_T$.puvProtocol.verify($K_{\text{authC}}, M, t$) = true
         then
  4:         $\pi^i_T$.canValidate $\leftarrow$ false
  5:         return $d$
  6: return rejected
```

Figure 12: CTAP 2.1 protocol functions. The current auth-C and validate-T functions are marked in red. Code below corresponds to CTAP 2.1+, where we highlight in blue the changes to the current version of the protocol: deriving two MAC keys instead of one and using them for bi-directional authentication. Additionally, we mark in orange our proposed CTAP 2.1++ modification of the token always sampling a *fresh* ECDH share.

The encrypt and decrypt functions both input a 4λ-bit key K and a message m or ciphertext c, then take the last 2λ bits as K_2, and use K_2 to respectively encrypt m or decrypt c, using the underlying encryption scheme SKE, instantiated as AES-256 in CBC-mode with random IV. authenticate inputs a key K, which can be 2λ or 4λ bits long (depending on whether the pinToken is the key or if a session key from Setup is used instead) and a message m, and uses the first 2λ bits as a key to authenticate m, producing a tag t using the underlying MAC. verify is almost identical, but also inputs a tag t and simply outputs if t is a valid tag for m or not. In both cases, the underlying MAC is HMAC-SHA-256 using a 256-bit key. The encapsulate and decapsulate functions perform the ECDH key exchange and then produce K_1 and K_2 by passing the ECDH result through a KDF (modeled by $\mathcal{H}_2$), instantiated as HKDF-SHA-256. In encapsulate, which is only executed by a client, both $K = (K_1, K_2)$ and the ECDH share c are output, while in decapsulate, which is only executed by the token, only K is returned.

PIN/UV Auth Protocol 1 is identical to PIN/UV Auth Protocol 2 in the initialize, getPublicKey and regenerate functions. The resetPuvToken is almost identical, but can sample a fresh pinToken that is λ-bit *or* 2λ-bit in length. encrypt and decrypt receive a 2λ-bit key K and use it directly to encrypt or decrypt the input m or c. SKE is instantiated as AES-256 in CBC-mode with *zero* IV. authenticate and verify use a key K (which can be the same key used for encrypt or decrypt) to authenticate a message m, producing a λ-bit tag t. In practice, t is the result of truncating the leftmost λ bits from the HMAC-SHA-256 2λ-bit output. Finally, encapsulate and decapsulate perform the same ECDH key exchange, but pass the ECDH shared secret through SHA-256 (modeled by $\mathcal{H}_1$) to produce a single, 2λ-bit symmetric key K.

Remark. While we generally maintained the CTAP 2.1 description as close as possible to the original work in [6], we do propose another modification to the description, which we also include in CTAP 2.1+. We include a new flag π_T^i.canValidate set to true in the protocol function obtainPinUvAuthToken-T. This flag determines if validate-T can verify the received tag or not for a given token session π_T^i. This modification to the protocol description aims at more closely following the CTAP 2.1 protocol specifications, which state that any command sent to a token which requires user presence (modeled by the bit d) and verifies correctly cause the token to revoke permissions from the currently in use binding state, effectively removing the possibility of validating more commands using the same binding state after the first successful validation.

CTAP 2.1+ description. For CTAP 2.1+, we propose a PIN/UV Auth Protocol which is identical to PIN/UV Auth Protocol 2 from CTAP 2.1 but with the addition of a new function expand, which inputs a 2λ-bit pinToken pt (binding state), produces a 4λ-bit expanded pinToken pt_e with a key derivation function modeled by $\mathcal{H}_3$ and then parses pt_e into two 2λ-bit keys, to be used for authenticating client-to-token and token-to-client messages. In practice, $\mathcal{H}_3$ could be instantiated with HKDF-SHA-256 called twice with two labels distinct from those in CTAP 2.1, similar to how (K_1, K_2) are derived, or simply instantiated with SHA-512. The CTAP 2.1+ protocol is identical to CTAP 2.1 until Bind is finished. The modifications can be seen in Figures 3, 12 and 10. After a client C and token T have finished Bind, C runs auth-C, which expands the binding state π_C^j.bs to produce a new authentication key K_{authC}, and uses K_{authC} to produce a tag t on message M. Afterwards, the token can validate (M, t) with the same binding state by running validate-T, which obtains K_{authC} and verifies (M, t). After validating (M, t) the token runs auth-T, which expands the binding state π_T^i.bs into a new authentication key K_{authT} (different from K_{authC}) and uses it to produce a new tag t on its response R. Finally, the client receives (R, t) and runs validate-C, which obtains K_{authT} and verifies (R, t). Notice that the reason we expand the binding state during authentication is to minimize as much as possible the modifications needed to achieve mutual authentication in CTAP 2.1+. This expansion allows us to ensure that every part of CTAP 2.1 up until the authentication of commands and responses is exactly the same, including the pinToken transmission from a token to the client.

CTAP 2.1++ description. CTAP 2.1++ is identical to CTAP 2.1+ with the exception of function obtainSharedSecret-T, where a fresh ECDH share is now always sampled by calling regenerate from the underlying PIN/UV Auth Protocol before obtaining it with getPublicKey.

C Preliminary Definitions

Definition 1. *Let $H : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ be a family of functions such that $H(k, \cdot) = H_k(\cdot)$ is efficient for all k. We say that H is collision resistant if the advantage of any efficient adversary $\mathcal{A}$ defined below is negligible.*

$$Adv_H^{coll}(\mathcal{A}) = Pr[k \xleftarrow{\$} \mathcal{K}, (m_1, m_2) \xleftarrow{\$} \mathcal{A}(k) : m_1 \neq m_2 \wedge H_k(m_1) = H_k(m_2)] .$$

Definition 2. *Let $\mathbb{G}$ be a cyclic group of prime order q and generator g. The Strong Computational Diffie-Hellman (sCDH) assumption states that given $a, b \xleftarrow{\$} \mathbb{Z}_q$, g, g^a, g^b, and an oracle O_a which, for any group elements $Y, Z \in \mathbb{G}$ checks if $Y^a = Z$, it is computationally infeasible for any efficient adversary $\mathcal{A}$ to compute g^{ab}. That is, the advantage of any efficient adversary $\mathcal{A}$ defined below is negligible.*

$$Adv_{\mathbb{G}}^{sCDH}(\mathcal{A}) = Pr[g^{ab} \leftarrow \mathcal{A}(\mathbb{G}, g, g^a, g^b, O_a)] .$$

Definition 3. *For any message authentication code $MAC = (Kg, Auth, Ver)$, we say MAC is SUF-CMA secure if for any efficient adversary $\mathcal{A}$ against the security experiment $Expt_{MAC}^{SUF\text{-}CMA}$ (Fig. 13), the advantage of $\mathcal{A}$ defined below is negligible.*

$$Adv_{MAC}^{SUF\text{-}CMA}(\mathcal{A}) = Pr[Expt_{MAC}^{SUF\text{-}CMA}(\mathcal{A}) = 1] .$$

```
$\mathrm{Expt}_{\mathrm{MAC}}^{\mathrm{SUF\text{-}CMA}}(\mathcal{A})$:
    $K \xleftarrow{\$} \mathrm{MAC.KG}()$
    $S \leftarrow \emptyset$
    $(m, t) \xleftarrow{\$} \mathcal{A}^{O_{\mathrm{Auth}}, O_{\mathrm{Ver}}}()$
    return $O_{\mathrm{Ver}}(m, t) \wedge (m, t) \notin S$

$O_{\mathbf{Auth}}(m)$:
    $t \leftarrow \mathrm{MAC}(K, m)$
    $S \leftarrow S \cup (m, t)$
    return $t$

$O_{\mathbf{Ver}}(m, t)$:
    $t' \leftarrow \mathrm{MAC}(K, m)$
    return $t' = t$
```

Figure 13: SUF-CMA Challenger and corresponding oracles O_{Auth} and O_{Ver}.

Definition 4. *Let $\mathcal{P}$ be a set of all keyed functions $F : \{0,1\}^l \times \{0,1\}^m \to \{0,1\}^m$, where F is bijective and there exists an efficient algorithm to compute $F(k, \cdot)$ and $F^{-1}(k, \cdot), \forall k \in \{0,1\}^l$. Let $\mathcal{F}$ be the set of all truly random permutations on $\{0,1\}^m$. Then, we say F is a pseudo-random permutation if, for all efficient adversary $\mathcal{A}$ and all $k \in \{0,1\}^l$, its advantage defined below is negligible.*

$$Adv_F^{prp}(\mathcal{A}) = Pr[F \xleftarrow{\$} \mathcal{P} : \mathcal{A}^{F(k,\cdot)}() = 1] - Pr[f \xleftarrow{\$} \mathcal{F} : \mathcal{A}^{f(\cdot)}() = 1] .$$

Definition 5. *For any symmetric encryption scheme $SKE = (Kg, Enc, Dec)$, we say SKE is IND-1\$PA secure if for any efficient adversary $\mathcal{A}$ against the security experiment $Expt_{SKE}^{IND\text{-}1\$PA}$ (Fig. 14), the advantage of $\mathcal{A}$ defined below is negligible.*

$$Adv_{SKE}^{IND\text{-}1\$PA}(\mathcal{A}) = \left| 2\, Pr[Expt_{SKE}^{IND\text{-}1\$PA}(\mathcal{A}) = 1] - 1 \right| .$$

Definition 6. *For any symmetric encryption scheme SKE = (Kg, Enc, Dec), we say SKE is IND-1\$PA-LPC secure if for any efficient adversary $\mathcal{A}$ against the security experiment $Expt_{SKE}^{IND\text{-}1\$PA\text{-}LPC}$ (Fig. 15), the advantage of $\mathcal{A}$ defined below is negligible.*

$$Adv_{SKE}^{IND\text{-}1\$PA\text{-}LPC}(\mathcal{A}) = \left| 2\ Pr[\,Expt_{SKE}^{IND\text{-}1\$PA\text{-}LPC}(\mathcal{A}) = 1] - 1 \right| .$$

Definition 7. *For any symmetric encryption scheme SKE = (Kg, Enc, Dec), we say SKE is IND-1\$PA-LHPC secure if for any efficient adversary $\mathcal{A}$ against the security experiment $Expt_{SKE}^{IND\text{-}1\$PA\text{-}LHPC}$ (Fig. 16), the advantage of $\mathcal{A}$ defined below is negligible.*

$$Adv_{SKE}^{IND\text{-}1\$PA\text{-}LHPC}(\mathcal{A}) = \left| 2\ Pr[\,Expt_{SKE}^{IND\text{-}1\$PA\text{-}LHPC}(\mathcal{A}) = 1] - 1 \right| .$$

Definition 8. *For any symmetric encryption scheme SKE = (Kg, Enc, Dec), we say SKE is IND-1\$PA-LHPC-H secure if for any efficient adversary $\mathcal{A}$ against the security experiment $Expt_{SKE}^{IND\text{-}1\$PA\text{-}LHPC\text{-}H}$ (Fig. 17), the advantage of $\mathcal{A}$ defined below is negligible.*

$$Adv_{SKE}^{IND\text{-}1\$PA\text{-}LHPC\text{-}H}(\mathcal{A}) = \left| 2\ Pr[\,Expt_{SKE}^{IND\text{-}1\$PA\text{-}LHPC\text{-}H}(\mathcal{A}) = 1] - 1 \right| .$$

```
$\mathsf{Expt}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA}}(\mathcal{A})$:
    $b \xleftarrow{\$} \{0,1\}$
    $K \xleftarrow{\$} \mathsf{SKE.KG}()$
    state $\leftarrow 0$
    $(m_0, m_1) \xleftarrow{\$} \mathcal{A}()$
    $c_{\mathsf{chal}} \xleftarrow{\$} O_{\mathsf{LR}}(m_0, m_1)$
    $b' \xleftarrow{\$} \mathcal{A}^{O_{\mathsf{LR}}, O_{\mathsf{RAND}}}(c_{\mathsf{chal}})$
    return $(b = b')$

$O_{\mathbf{LR}}(m_0, m_1)$:
    if state $\neq 0$ then
        return $\perp$
    if $|m_0| \neq |m_1|$ then
        return $\perp$
    $c_{\mathsf{chal}} \xleftarrow{\$} \mathsf{SKE.Enc}(K, m_b)$
    state $\leftarrow 1$
    return $c_{\mathsf{chal}}$

$O_{\mathbf{RAND}}(l)$:
    $m_0', m_1' \xleftarrow{\$} \{0,1\}^l$
    $c' \xleftarrow{\$} \mathsf{SKE.Enc}(K, m_b')$
    return $(m_0', m_1', c')$
```

Figure 14: IND-1\$PA Challenger, and corresponding Left-Right oracle $O_{\mathbf{LR}}$ and RAND oracle $O_{\mathbf{RAND}}$.

```
$\mathsf{Expt}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LPC}}(\mathcal{A})$:
    $b \xleftarrow{\$} \{0,1\}$
    $K \xleftarrow{\$} \mathsf{SKE.KG}()$
    state $\leftarrow 0$
    $(m_0, m_1) \xleftarrow{\$} \mathcal{A}()$
    $c_{\mathsf{chal}} \xleftarrow{\$} O_{\mathsf{LR}}(m_0, m_1)$
    $b' \xleftarrow{\$} \mathcal{A}^{O_{\mathsf{LR}}, O_{\mathsf{RAND}}, O_{\mathsf{LPC}}}(c_{\mathsf{chal}})$
    return $(b = b')$

$O_{\mathbf{RAND}}(l)$:
    $m_0', m_1' \xleftarrow{\$} \{0,1\}^l$
    $c' \xleftarrow{\$} \mathsf{SKE.Enc}(K, m_b')$
    return $(m_0', m_1', c')$

$O_{\mathbf{LR}}(m_0, m_1)$:
    if state $\neq 0$ then
        return $\perp$
    if $|m_0| \neq |m_1|$ then
        return $\perp$
    $c_{\mathsf{chal}} \xleftarrow{\$} \mathsf{SKE.Enc}(K, m_b)$
    state $\leftarrow 1$
    return $c_{\mathsf{chal}}$

$O_{\mathbf{LPC}}(c)$:
    if $c = c_{\mathsf{chal}}$ then
        return $\perp$
    return $m_0 = \mathsf{SKE.Dec}(K, c)$
```

Figure 15: IND-1\$PA-LPC Challenger, and corresponding Left-Right oracle $O_{\mathbf{LR}}$, RAND oracle $O_{\mathbf{RAND}}$ and LPC oracle $O_{\mathbf{LPC}}$.

```
$\mathsf{Expt}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LHPC}}(\mathcal{A})$:
    $b \xleftarrow{\$} \{0,1\}$
    $K \xleftarrow{\$} \mathsf{SKE.KG}()$
    state $\leftarrow 0$
    $(m_0, m_1) \xleftarrow{\$} \mathcal{A}()$
    $c_{\mathsf{chal}} \xleftarrow{\$} O_{\mathsf{LR}}(m_0, m_1)$
    $b' \xleftarrow{\$} \mathcal{A}^{O_{\mathsf{LR}}, O_{\mathsf{RAND}}, O_{\mathsf{LHPC}}}(c_{\mathsf{chal}})$
    return $(b = b')$

$O_{\mathbf{RAND}}(l)$:
    $m_0', m_1' \xleftarrow{\$} \{0,1\}^l$
    $c' \xleftarrow{\$} \mathsf{SKE.Enc}(K, m_b')$
    return $(m_0', m_1', c')$

$O_{\mathbf{LR}}(m_0, m_1)$:
    if state $\neq 0$ then
        return $\perp$
    if $|m_0| \neq |m_1|$ then
        return $\perp$
    $c_{\mathsf{chal}} \xleftarrow{\$} \mathsf{SKE.Enc}(K, m_b)$
    state $\leftarrow 1$
    return $c_{\mathsf{chal}}$

$O_{\mathbf{LHPC}}(c)$:
    if $c = c_{\mathsf{chal}}$ then
        return $\perp$
    return $\mathsf{H}(m_0) = \mathsf{SKE.Dec}(K, c)$
```

Figure 16: IND-1\$PA-LHPC Challenger, and corresponding Left-Right oracle $O_{\mathbf{LR}}$, RAND oracle $O_{\mathbf{RAND}}$ and LHPC oracle $O_{\mathbf{LHPC}}$.

```
$\mathsf{Expt}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LHPC\text{-}H}}(\mathcal{A})$:
    $b \xleftarrow{\$} \{0,1\}$
    $K \xleftarrow{\$} \mathsf{SKE.KG}()$
    state $\leftarrow 0$
    $(m_0, m_1) \xleftarrow{\$} \mathcal{A}()$
    $c_{\mathsf{chal}} \xleftarrow{\$} O_{\mathsf{LR}}(m_0, m_1)$
    $t_{\mathsf{chal}} \xleftarrow{\$} \mathsf{H}(K, c_{\mathsf{chal}})$
    $b' \xleftarrow{\$} \mathcal{A}^{O_{\mathsf{LR}}, O_{\mathsf{RAND}}, O_{\mathsf{LHPC}}}(c_{\mathsf{chal}}, t_{\mathsf{chal}})$
    return $(b = b')$

$O_{\mathbf{RAND}}(l)$:
    $m_0', m_1' \xleftarrow{\$} \{0,1\}^l$
    $c' \xleftarrow{\$} \mathsf{SKE.Enc}(K, m_b')$
    return $(m_0', m_1', c')$

$O_{\mathbf{LR}}(m_0, m_1)$:
    if state $\neq 0$ then
        return $\perp$
    if $|m_0| \neq |m_1|$ then
        return $\perp$
    $c_{\mathsf{chal}} \xleftarrow{\$} \mathsf{SKE.Enc}(K, m_b)$
    state $\leftarrow 1$
    return $c_{\mathsf{chal}}$

$O_{\mathbf{LHPC}}(c)$:
    if $c = c_{\mathsf{chal}}$ then
        return $\perp$
    return $\mathsf{H}(m_0) = \mathsf{SKE.Dec}(K, c)$
```

Figure 17: IND-1\$PA-LHPC-H Challenger, and corresponding Left-Right oracle $O_{\mathbf{LR}}$, RAND oracle $O_{\mathbf{RAND}}$ and LHPC oracle $O_{\mathbf{LHPC}}$.

We present the full proof for the IND-1\$PA-LHPC security of CBC in the full version [2], assuming AES is a pseudo-random permutation.

D Security Proofs

Our proofs follow the *game-based technique* (see [23] for a tutorial) and some proofs require a *hybrid argument* (see [15] for a tutorial).

D.1 Proof of Theorem 1

Before showing the proof of this theorem, we note that it can be expressed differently if the adversary's queries are counted in a different way, which may lead to more relevant bounds for different threat models. As expected, the adversary has negligible advantage, except if it actively interacts with a token via **Send-Bind-T** enough times to guess a user pin. As stated, the theorem captures an adversary that is not constrained in its number of attempts, except by an arbitrary upper bound $q_{\mathsf{Send}}^{\mathsf{act}}$. In practice, however, the attacker may have more restrictions and our bound can be adapted to reflect them, as we remark at the end of the proof.

Proof. *(Sketch.)* Let Pr_i denote the probability that Game i outputs 1.

Game 0. This is the original experiment. Therefore,

$$\mathsf{Pr}_0 = \mathsf{Adv}^{\mathsf{SUF\text{-}t}}_{\mathsf{CTAP\ 2.1}}(\mathcal{A})\ .$$

GAME 1. In this game, we replace every symmetric session key K_1 and K_2 calculated in encapsulate and decapsulate from Fig. 10 as $K_1 \leftarrow \mathcal{H}_2(Z,\cdot)$ and $K_2 \leftarrow \mathcal{H}_2(Z,\cdot)$, where Z is the ECDH shared secret derived by a client session π^j_C and token session π^i_T in a **Setup** or **Execute** query, with independent random values $\tilde{K_1}$ and $\tilde{K_2}$. We define bad as the event that, at some point during the game, π^j_C and π^i_T are involved in a **Setup** or **Execute** and $\mathcal{A}$ queries $\mathcal{H}_2$ with Z, and bound the probability of bad by constructing an adversary $\mathcal{B}_1$ against the sCDH security of the underlying ECDH group, such that if $\mathcal{A}$ causes bad, $\mathcal{B}_1$ wins the sCDH game. Since token public keys are only given to $\mathcal{A}$ in **Setup**, **Execute** and **Send-Bind-T** oracles, we have $|\mathsf{Pr}_0 - \mathsf{Pr}_1| \leq (q_{\mathsf{S}} + q_{\mathsf{E}} + q_{\mathsf{Send}})\ \mathsf{Adv}^{\mathsf{sCDH}}_{\mathsf{ECDH}}(\mathcal{B}_1)$.

GAME 2. In this game, in every **Setup** query, the pin is no longer stored in the token state st_T, instead being stored in a new strucuture accessible only by the challenger. Since this model never allows $\mathcal{A}$ to corrupt any token, GAMES 1 and 2 are functionally identical. Therefore, $\mathsf{Pr}_1 = \mathsf{Pr}_2$.

GAME 3. In this game, $\mathcal{A}$ immediately loses if the challenger samples two identical ECDH public keys from a client or token. Given that client ECDH public keys are sampled only in **Setup** and **Execute** queries, and token ECDH public keys are sampled in **NewT**, **Reboot** and **Send-Bind-T** queries, we have $|\mathsf{Pr}_2 - \mathsf{Pr}_3| \leq (q_{\mathsf{S}} + q_{\mathsf{E}} + q_{\mathsf{NT}} + q_{\mathsf{R}} + 2q_{\mathsf{Send}})^2\ /\ (2q)$.

GAME 4. In this game, $\mathcal{A}$ loses if a collision occurs on some output of H when called by the challenger. Since H is assumed to be collision resistant, there exists an adversary $\mathcal{B}_4$ such that $|\mathsf{Pr}_3 - \mathsf{Pr}_4| \leq \mathsf{Adv}^{\mathsf{coll}}_{\mathsf{H}}(\mathcal{B}_4)$.

GAME 5. This game replaces every encrypted pin in **Setup** queries with an encrypted constant pin 0000. Additionally, if the ECDH client and token public keys used in **Setup** are used again in **Send-Bind-T**, the token encrypts and outputs a random pinToken $\tilde{pt}$ while setting the real pinToken as its binding state. We bound the advantage of $\mathcal{A}$ against GAME 5 by reduction to the IND-1\$PA-LHPC security of the underlying encryption scheme SKE. Therefore, we construct an adversary $\mathcal{B}_5$ against the IND-1\$PA-LHPC security of SKE such that $|\mathsf{Pr}_4 - \mathsf{Pr}_5| \leq q_{\mathsf{S}}\ \mathsf{Adv}^{\mathsf{IND\text{-}1\$PA\text{-}LHPC}}_{\mathsf{SKE}}(\mathcal{B}_5)$.

GAME 6. This game replaces every encrypted hashed pin H(pin) and pinToken pt in **Execute** queries with an encrypted hashed constant H(0000) and a random pinToken $\tilde{pt}$ respectively. We bound the advantage of $\mathcal{A}$ against GAME 6 by reduction to the IND-1\$PA-LPC security of the underlying encryption scheme SKE. Therefore, we construct an adversary $\mathcal{B}_6$ against the IND-1\$PA-LPC security of SKE such that $|\mathsf{Pr}_5 - \mathsf{Pr}_6| \leq q_{\mathsf{E}}\ \mathsf{Adv}^{\mathsf{IND\text{-}1\$PA\text{-}LPC}}_{\mathsf{SKE}}(\mathcal{B}_6)$.

GAME 7. In this game, user pins are no longer sampled in **NewU**, being sampled instead only when relevant to answer to some query from $\mathcal{A}$, which happens only in **Corrupt** and **Send-Bind-T** queries. Since every pin is independently sampled from $\mathcal{D}$, it is independent from anything that happens during the experiment, and thus GAMES 6 and 7 are functionally identical. Therefore, $\mathsf{Pr}_6 = \mathsf{Pr}_7$.

GAME 8. In this game, the challenger rejects any attempt from $\mathcal{A}$ to actively guess the correct pin, unless the pin for the target token has been corrupted. By considering $q^{\mathsf{act}}_{\mathsf{Send}}$ the maximum number of active queries to **Send-Bind-T**, where $\mathcal{A}$ actively attempts at guessing the pin, and $1/2^{h_{\mathcal{D}}}$ as the maximum probability that $\mathcal{A}$ can guess the pin in any query, we have $|\mathsf{Pr}_7 - \mathsf{Pr}_8| \leq q^{\mathsf{act}}_{\mathsf{Send}}/2^{h_{\mathcal{D}}}$.

GAME 9. In this game, $\mathcal{A}$ loses if two pinTokens that are sent to the adversary in either an **Execute** or **Send-Bind-T** query collide. Since, at most, one pinToken is generated and given to $\mathcal{A}$ in each **Execute** or **Send-Bind-T** query, we have $|\mathsf{Pr}_8 - \mathsf{Pr}_9| \leq (q_{\mathsf{E}} + q_{\mathsf{Send}})^2\ /\ 2^{2\lambda+1}$.

GAME 10. In this game, when $\mathcal{A}$ queries **Validate-T** with message M and a tag t on a token session π^i_T that finished Bind passively (which happens *only* in **Execute** and passive **Send-Bind-T** queries), such that (M, t) would constitute a valid forgery, the challenger still rejects (recall that $\mathcal{A}$ cannot win against any token session π^i_T that was actively attacked in **Send-Bind-T** queries, since after GAME 8 this means T's pin must have been corrupted). We consider this bad event and bound the advantage of $\mathcal{A}$ by reducing to the SUF-CMA security of the underlying MAC, by constructing an efficient adversary $\mathcal{B}_{10}$ against the SUF-CMA security of MAC such that if bad happens, $\mathcal{B}_{10}$ wins the SUF-CMA game. Therefore, we have $|\mathsf{Pr}_9 - \mathsf{Pr}_{10}| \leq (q_{\mathsf{E}} + q_{\mathsf{Send}})\ \mathsf{Adv}^{\mathsf{SUF\text{-}CMA}}_{\mathsf{MAC}}(\mathcal{B}_{10})$.

FINAL ANALYSIS. After the modifications in GAME 10, $\mathcal{A}$ can no longer win. Indeed, for $\mathcal{A}$ to win, it must satisfy at least one of the four conditions in Token-Win-SUF-t. The first condition is always false, since $d = 1$ must always be true for any **Validate-T** query to accept a tag t. The second and third conditions are also always false. Collisions between sid values in client sessions have been ruled out in GAME 3, by avoiding client public key collisions, while collisions between sid values in token sessions have been ruled out in GAME 9, by removing collisions between any pinToken that is encrypted and sent to $\mathcal{A}$. Therefore, it can never be the case that two different client or token sessions have the same sid. Finally, $\mathcal{A}$ can never win via the last condition, because in GAME 10 every valid forgery attempt against any token session via a **Validate-T** query is rejected by the challenger. Therefore, it is always true that $\mathsf{Pr}_{10} = 0$. Note that the running times of all adversaries are close to that of $\mathcal{A}$. □

Remark. In GAME 8, we bound $\mathcal{A}$'s advantage as a function of $q^{\mathsf{act}}_{\mathsf{Send}}$, i.e., how many active attacks $\mathcal{A}$ makes against *any* token. This captures the worst case scenario where $\mathcal{A}$ can always reset pinRetries by executing a passive Bind for *any* token (i.e., the adversary may have access to any user from any token to input the correct pin). However, it is also reasonable to consider the case where $\mathcal{A}$ has access to an arbitrary number of tokens but not to their users, which limits the amount of active attacks on any token to pinRetriesMax. Considering $q^{\mathsf{act}}_{\mathsf{NT}}$ as the number of tokens created that are actively attacked by $\mathcal{A}$ during the experiment, we could write the bound as $|\mathsf{Pr}_7 - \mathsf{Pr}_8| \leq \mathsf{pinRetriesMax} \cdot q^{\mathsf{act}}_{\mathsf{NT}}/2^{h_{\mathcal{D}}}$.

D.2 Proof of Theorem 2

The proof of security of CTAP 2.1+ is identical to the proof of CTAP 2.1 (as sketched in Appendix D.1) until GAME 9. We summarize the modifications and extra steps next.

GAME 10. In this game, $\mathcal{A}$ loses if it queries random oracle $\mathcal{H}_3$ with any pinToken generated throughout the experiment and set as the binding state of a token or client session, but for which the adversary has no information. These pinTokens correspond to sessions where the adversary behaved passively, which includes all of the Execute queries, and possibly some of the Send-Bind-T queries. The two games are identical until bad, and so we have $|\Pr_9 - \Pr_{10}| \leq q_{\mathcal{H}_3}\ (q_{\mathsf{E}} + q_{\mathsf{Send}})/2^{2\lambda}$. After this game, the MAC keys used by these sessions to authenticate commands are information theoretically hidden from the adversary.

Game 11. In this game, when $\mathcal{A}$ queries Validate-T with message M and a tag t on a token session π_T^i that finished Bind passively (which happens *only* in Execute and passive Send-Bind-T queries), such that (M, t) would constitute a valid forgery, the challenger still rejects (recall that $\mathcal{A}$ cannot win against any token session π_T^i that was actively attacked in Send-Bind-T queries, since after Game 8 this means T's pin must have been corrupted). We consider this bad event and bound the advantage of $\mathcal{A}$ by reducing to the SUF-CMA security of the underlying MAC, by constructing an efficient adversary $\mathcal{B}_{11}$ against the SUF-CMA security of MAC such that if bad happens, $\mathcal{B}_{11}$ wins the SUF-CMA game. Therefore, we have $|\Pr_{10} - \Pr_{11}| \leq (q_{\mathsf{E}} + q_{\mathsf{Send}})\ \mathsf{Adv}_{\mathsf{MAC}}^{\mathsf{SUF\text{-}CMA}}(\mathcal{B}_{11})$.

Game 12. In this game, when $\mathcal{A}$ queries Validate-C with message M and a tag t and a client session π_C^j that finished Bind (which happens only in Execute queries), such that (M, t) would constitute a valid forgery, the challenger still rejects. Much like in Game 11, we bound the advantage of $\mathcal{A}$ against Game 11 via reduction to the SUF-CMA security of MAC. Therefore, there exists an efficient adversary $\mathcal{B}_{12}$ such that $|\Pr_{11} - \Pr_{12}| \leq q_{\mathsf{E}}\ \mathsf{Adv}_{\mathsf{MAC}}^{\mathsf{SUF\text{-}CMA}}(\mathcal{B}_{12})$.

Final analysis. After the modifications in Game 12, $\mathcal{A}$ can no longer win. Indeed, for $\mathcal{A}$ to win, it must satisfy at least one of the four conditions in Token-Win-SUF-t or one of the three conditions in Client-Win-SUF-t. The reasoning for $\mathcal{A}$ not being able to win via Token-Win-SUF-t is identical to the reasoning used for the previous CTAP 2.1 security proof. Additionally, $\mathcal{A}$ cannot win Game 12 via the conditions from Client-Win-SUF-t. Indeed, the first and second conditions are identical to the second and third conditions from Token-Win-SUF-t, and are therefore always false, since no collisions between client session sid values and token session sid values can occur after Games 3 and 9 respectively. $\mathcal{A}$ can also never win via the third condition in Client-Win-SUF-t, because in Game 12 every valid forgery attempt against any client session via Validate-C query is rejected by the challenger. Therefore, it is always true that $\Pr_{12} = 0$. Note that the running times of all adversaries are close to that of $\mathcal{A}$.

D.3 Proof of Theorem 4

Let $\mathbf{G}_0$ be the original experiment $\mathsf{Expt}_{\mathsf{CTAP\ 2.1++}}^{\mathsf{priv}}(\mathcal{A})$. We consider a series of hybrid games analogous to the proof of Theorem 2 (authentication of CTAP 2.1+). To save space, we briefly mention what each hybrid does without providing details. $\mathbf{G}_1$ replaces every symmetric session key K_1 and K_2 with independent random keys. $\mathbf{G}_2$ will store PIN in a new structure instead of token state. $\mathbf{G}_3$ will remove collisions in sampled ECDH shares. $\mathbf{G}_4$ removes hash collisions of H. $\mathbf{G}_5$ will replace encrypted PIN with encrypted constant PIN 0000. In $\mathbf{G}_6$, we switch encryption of hashed PIN and encryption of pintoken to be encryption of H(0000) and encryption of random pintoken during Execute in Phase 1. In $\mathbf{G}_7$, we will sample PIN only when adversary queries CorruptUser or Send-Bind-T. In $\mathbf{G}_8$, we will reject attempts from $\mathcal{A}$ to actively guess the PIN through Send-Bind-T operations.

Suppose $\mathcal{A}$ that makes at most $q_{\mathsf{S}}, q_{\mathsf{E}}, q_{\mathsf{Send}}, q_{\mathsf{NT}}$ and q_{R} queries to Setup, Execute, Send-Bind-T, NewT and Reboot, and at most $q_{\mathsf{Send}}^{\mathsf{act}}$ active queries to Send-Bind-T. Then there exist efficient adversaries $\mathcal{B}_1$, $\mathcal{B}_4$, $\mathcal{B}_5$, $\mathcal{B}_6$ such that the following claims hold: (where $\Pr_i$ denote the probability that $\mathbf{G}_i$ outputs 1)

$$\Pr_0 = \mathsf{Adv}_{\mathsf{CTAP\ 2.1++}}^{\mathsf{priv}}(\mathcal{A})/2 + 1/2 \tag{1}$$

$$\begin{aligned}\Pr_0 - \Pr_8 \leq\ & (q_{\mathsf{S}} + q_{\mathsf{E}} + q_{\mathsf{Send}})\ \mathsf{Adv}_{\mathsf{ECDH}}^{\mathsf{sCDH}}(\mathcal{B}_1) \\ & + (q_{\mathsf{S}} + q_{\mathsf{E}} + q_{\mathsf{NT}} + q_{\mathsf{R}} + 2q_{\mathsf{Send}})^2\ /\ (2q) \\ & + \mathsf{Adv}_{\mathsf{H}}^{\mathsf{coll}}(\mathcal{B}_4) \\ & + q_{\mathsf{S}}\ \mathsf{Adv}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LHPC}}(\mathcal{B}_5) \\ & + q_{\mathsf{E}}\ \mathsf{Adv}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LPC}}(\mathcal{B}_6) \\ & + q_{\mathsf{Send}}^{\mathsf{act}}/2^{h_{\mathcal{D}}}\end{aligned} \tag{2}$$

$$\Pr_8 = 1/2 \tag{3}$$

The above claims are justified as follows. Equation (1) is by definition of $\mathsf{Adv}_{\mathsf{CTAP\ 2.1++}}^{\mathsf{priv}}(\mathcal{A})$. Inequality (2) follows from the mPACA security proof of CTAP 2.1+ (proof of Theorem 2 in Section 4.4).

To justify equality (3), we note that adversary's view includes public information of different tokens and communications within each session. In $\mathsf{Expt}_{\mathsf{CTAP\ 2.1++}}^{\mathsf{priv}}(\mathcal{A})$, we require that two challanged token's users cannot be corrupted. Therefore, by $\mathbf{G}_8$, we have swapped encryption of pin and pin hashes to be encryption of 0000 and H(0000). Diffe-Hellman shares are also freshly generated. Therefore, the attacker's view is almost independent of the actual bit b. Special care is needed to prevent trivial attacks. Suppose b is 0, and attacker makes a LEFT query. If the attacker queries regular query T_0 on the same index, token T_0 will reject (while T_1 will accept). If the LEFT query is bind, and attacker queries regular Auth/Validate on T_0 on the same index, it will accept (while T_1 will reject). If the LEFT query is Setup, and attacker makes regular queries on T_0, it will accept (while T_1 will reject). If attacker inputs a wrong PIN in LEFT query, the pinRetry of T_0 will decrease by 1. We addresses these subtleties specifically and carefully by introducing different checks. Therefore, either the attacker has broken these checks, which automatically fails (the game will return a random bit, which gives attacker advantge 0), or it can only guess the challenge bit with probability 1/2.

D.4 Proof of Theorem 5

Let $\Pr_i$ be the probability that Game i outputs 1.

Game 0. This is the original composed privacy experiment $\mathsf{Expt}_{\mathsf{PIA+mPACA}}^{\mathsf{com\text{-}priv}}$ when the underlying challenge bit $b = 0$.

Game 1. This game is identical to Game 0 except that in aResp, the oracle will always add cid to $\mathcal{L}_{\mathsf{ch}}^{\mathsf{a}}$ and add (T, j') to $\mathcal{L}_{\mathsf{ch}}^{\mathsf{bd}}$. We can introduce a new flag bad, and set bad$\leftarrow$ true when tokenBindPartner(T, j') is $\perp$, or if the binding partner is (C, j),$(C, j, m_{\mathsf{acom}}, t_{\mathsf{cl}}) \notin\in \mathcal{L}_{\mathsf{authC}}$, which means message, tag pair $(m_{\mathsf{acom}}, t_{\mathsf{cl}})$ is not output by client session π_C^j.

Now, $\Pr_0 - \Pr_1 = \Pr[\mathsf{bad} \leftarrow \mathsf{true}]$. We claim that there exists a mPACA authentication adversary $\mathcal{B}_1$ such that $\Pr[\mathsf{bad} \leftarrow \mathsf{true}] \leq$

$\mathsf{Adv}^{\mathsf{SUF\text{-}t}}_{\mathsf{mPACA}}(\mathcal{B}_1)$. $\mathcal{B}_1$ samples bit b, and simulates mPACA oracles using its own oracles provided in mPACA authentication experiment. $\mathcal{B}_1$ then internally initializes PlA oracles and LEFT and RIGHT challenge oracles. In Game 1, if bad is true, then composed unlinkability adversary $\mathcal{A}$ has let mPACA token session (T, j') to accept a command where either (T, j') does not have a binding partner, or the command is not authorized by the partner. Both cases are winning conditions for mPACA authentication experiment. Since $\mathcal{B}_1$ forwards all mPACA operations to its own oracles, it will trigger the winner conditions in Token-Win-Auth in mPACA authentication experiment. Therefore, $\Pr[\mathsf{bad} \leftarrow \mathsf{true} \leq \mathsf{Adv}^{\mathsf{SUF\text{-}t}}_{\mathsf{mPACA}}(\mathcal{B}_1)$.

Game 2. Game 2 is identical to Game 1, except that two helper functions rResp' and aResp' that are used in r/aLEFT and r/aRIGHT, will take in both tokens T_0 and T_1 (the order is permuted depending on whether it is called inside LEFT or RIGHT). rResp' and aResp' will use the first token to perform authorization, just like Game 1, but will use the other token to get PlA response through rRsp or aRsp. Essentially, LEFT (RIGHT) oracle will now use Token T_1's (T_0's) PlA response, but authorizing it using T_0's(T_1's) binding state.

We claim that there exists a PlA privacy adversary $\mathcal{B}_2$ such that $|\Pr_1 - \Pr_2| \leq \mathsf{Adv}^{\mathsf{priv}}_{\mathsf{PlA}}(\mathcal{B}_2)$. We construct $\mathcal{B}_2$ such that it queries its own PlA oracles to get token responses, and will sample corresponding mPACA instances to simulate the rest of mPACA queries. In the PlA privacy game that $\mathcal{B}_2$ is playing, when the underlying bit b_{PlA} is 0, $\mathcal{B}_2$ is simulating Game 0; while when b_{PlA} is 1, C is simulating Game 1.

Game 3. This is the original composed privacy game $\mathsf{Expt}^{\mathsf{com\text{-}priv}}_{\mathsf{PlA+mPACA}}$ when the underlying challenge bit $b = 1$.

We claim that there exists an mPACA privacy adversary $\mathcal{B}_3$ such that $|\Pr_2 - \Pr_3| \leq \mathsf{Adv}^{\mathsf{priv}}_{\mathsf{mPACA}}(\mathcal{B}_3)$. We construct $\mathcal{B}_3$ such that it simply queries its own oracle provided in the mPACA privacy experiment for all mPACA queries. For queries to the PlA oracles, $\mathcal{B}_3$ simulate PlA oracles, and always use T_1's PlA response in LEFT, and T_0's PlA response in RIGHT, just like Game 2. Now, in the mPACA privacy game that $\mathcal{B}_3$ is playing, when the underlying bit b_{mPACA} is 0, $\mathcal{B}$ is simulating Game 2; while when b_{mPACA} is 1, $\mathcal{B}$ is simulating Game 3.

Final analysis. The proof is concluded as follows:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{com\text{-}priv}}_{\mathsf{PlA+mPACA}}(\mathcal{A}) &= |\Pr_0 - \Pr_3| \\ &\leq |\Pr_0 - \Pr_1| + |\Pr_1 - \Pr_2| + |\Pr_2 - \Pr_3| \\ &\leq \mathsf{Adv}^{\mathsf{SUF\text{-}t}}_{\mathsf{mPACA}}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{priv}}_{\mathsf{mPACA}}(\mathcal{B}_2) \\ &\quad + \mathsf{Adv}^{\mathsf{priv}}_{\mathsf{PlA}}(\mathcal{B}_3)\ .\end{aligned}$$

E CTAP 2.1 Security for PIN/UV Auth Protocol 1

We present a brief overview of the CTAP 2.1 security proof when instantiated with PIN/UV Auth Protocol 1, by outlining its differences from the CTAP 2.1 proof when using PIN/UV Auth Protocol 2.

Most games from the CTAP 2.1 security proof that was presented in Section 4 remain unchanged, since they are not affected from which protocol is instantiated. Therefore, we focus only on Games 5, 9 and 10, which differ when CTAP 2.1 is instantiated with PIN/UV Auth Protocol 1.

Game 5. This game replaces all encrypted pins on Setup with constant values, and then also replaces all pinTokens sent during Bind with random values only when using the same symmetric key used on Setup. When considering PIN/UV Auth Protocol 1, we can no longer reduce to the IND-1\$PA-LHPC security of the underlying encryption scheme SKE, because during Setup the same symmetric key K is used to encrypt $\mathsf{H}(0000)$ into c_p and then to authenticate c_p by producing a tag t_p. Therefore, we must reduce to a variant of this security definition, which we call IND-1\$PA-LHPC-H.

Game 9. This game eliminates collisions between pinTokens that are sent to $\mathcal{A}$ in Execute or Send-Bind-T queries. When considering PIN/UV Auth Protocol 1, each pinToken has size $\mu\lambda$, for $\mu \in \{1, 2\}$, which means the upper bound for the probability that a collision between two pinTokens occurs is $1/\lambda$. Therefore, we have $|\Pr_8 - \Pr_9| \leq (q_{\mathsf{E}} + q_{\mathsf{Send}})^2/2^{\lambda+1}$.

Game 10. In this game, each tag t corresponds to the leftmost λ bits from the 2λ-bit output of MAC, instantiated as HMAC-SHA-256. The reduction is the same, under the assumption that the truncated output of HMAC-SHA-256 is still a secure MAC.

F Proof Shortcomings in Nina et al. [6]

We show the shortcomings of the CTAP 2.1 proof in Nina *et al.* [6] as follows.

Active binding attacks against clients are too strong. We present here a more detailed explanation of the attack that an adversary can perform during Bind if allowed to be active when delivering the final message containing the encrypted pinToken from the token to the client.

CTAP 2.1 with PIN/UV Auth Protocol 2 uses AES-256 in CBC mode with random IV to encrypt the 128-bit pinHash that is sent from a client to a token during Bind and, immediately after that, to encrypt the 256-bit pinToken back to the client. Crucially, both encryptions use the same symmetric key (this is true for both PIN/UV Auth Protocol versions). Consider a ciphertext $c_{ph} = \mathsf{IV} \,||\, \mathsf{AES}(\mathsf{IV} \oplus \mathsf{pinHash})$, generated by a client session π^j_C. If the hash of the PIN checks out, π^i_T will accept and output a ciphertext c_{pt}: this is a CBC encryption with a fresh IV and two blocks encoding a pinToken to be decrypted by π^j_C. Now, $\mathcal{A}$ can create a new ciphertext $\tilde{c}_{pt} = \mathsf{IV} \,||\, \mathsf{AES}(\mathsf{IV} \oplus \mathsf{pinHash}) \,||\, \mathsf{AES}(\mathsf{IV} \oplus \mathsf{pinHash})$ and deliver $\tilde{c}_{pt}$ to π^j_C. Since AES in CBC mode is not an authenticated encryption scheme, π^j_C always decrypts $\tilde{c}_{pt}$, recovering a mauled pinToken of the form: $\pi^j_C.\mathsf{bs} = \mathsf{pinHash} \,||\, \mathsf{IV} \oplus \mathsf{pinHash} \oplus \mathsf{AES}(\mathsf{IV} \oplus \mathsf{pinHash})$. Formally, and this is where the proof in [6] is incorrect, this means that the recovered pinToken can actually depend on the user's PIN. In practice, this means that when the client issues a command authenticated with this mauled pinToken, producing a tag t, the adversary can use the MAC verification algorithm to perform an offline dictionary attack, by hashing values in the PIN space and checking if they produce an identical tag on the same command issued by the client. This is possible because the pinHash is the only part of the mauled pinToken the adversary does not know. This will allow the adversary to win the game with probability close to 1, and so there is no hope of proving CTAP 2.1 secure in such a model.

IND-1CPA is not enough. We also identified a minor oversight in the proof in [6] (Appendix I, Game 12) that applies only to unlikely cases where tokens do not refresh their DH shares after Setup. We explain this next.

The reduction to the security of the symmetric encryption scheme used to communicate with tokens has been improved in [6] compared to [3]. In particular, it was shown that, during Bind, a stronger (plaintext-checking) assumption was needed to deal with active attacks on the token. However, this was considered to be unnecessary during Setup, which is not the case if DH shares are *reused* by the token. Indeed, while it is true that a query to Setup always results in a new Diffie-Hellman key exchange between π_C^j and π_T^i, it cannot be said, even when excluding collisions between DH shares and symmetric keys, that the symmetric key used to encrypt the pin sent to π_T^i will not be used again later in the experiment. Since every Setup trace is given to $\mathcal{A}$, the adversary can attempt to start a Bind run with another token session $\pi_T^{i'}$ of the same token, using a DH share that was previously used by π_C^j during Setup. If T did not regenerate its DH share, then it will derive the same symmetric key that was used in Setup, and therefore might encrypt a pinToken with the same key used to encrypt the pin in Setup. Therefore, the encryption scheme that encrypts the user pin during Setup must be $\mathsf{IND\text{-}1\$PA\text{-}LHPC}$ secure (defined in Appendix C), rather than only $\mathsf{IND\text{-}1CPA}$. This is a minor change to the proof, which we handle, and it will not be necessary if the token is guaranteed to reset its state (via Reboot) after Setup.

G PlA Models and WebAuthn Analysis

G.1 PlA Protocol Syntax

We closely follow [6, 7] to define the syntax for *passwordless authentication (PlA)* protocols. Our syntax is very similar to the ePlA protocol defined in [6] and the ePlAA protocol defined in [7]; we simply call our PlA primitive a PlA protocol.

A PlA protocol PlA consists of two phases Register and Authenticate:

Register: a two-pass challenge-response protocol run among a token T, a client C, and a server S, which is run at most once per tuple (T, S). At the end of Register, both T and S hold registration contexts, which are relevant for subsequent authentications. Register can be decomposed into the following algorithms:

- $m_{\mathsf{rch}} \xleftarrow{\$} \mathsf{rChal}(S, \mathsf{tb}, \mathsf{UV})$: inputs a server S,[9] a token binding state tb, and a user verification condition $\mathsf{UV} \in \{\mathsf{T}, \mathsf{F}\}$, and outputs a challenge message m_{rch}. It does not change the state of the server S.
- $(m_{\mathsf{rcl}}, m_{\mathsf{rcom}}) \xleftarrow{\$} \mathsf{rCom}(\mathsf{id}_S, m_{\mathsf{rch}}, \mathsf{tb})$: run by the stateless client; it inputs the intended server identity id_S, a challenge message m_{rch}, and a token binding state tb, and outputs a client message m_{rcl} and a command message m_{rcom}.
- $(m_{\mathsf{rrsp}}, \mathsf{rc}_T, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \xleftarrow{\$} \mathsf{rRsp}(T, m_{\mathsf{rcom}})$: inputs a token T and a command message m_{rcom} and outputs a response message m_{rrsp}, the token-side registration context rc_T, a credential identifier cid, a session identifier sid, and agreed contents agCon from the perspective of the token T.
- $(b, \mathsf{rc}_S, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \xleftarrow{\$} \mathsf{rVrfy}(S, m_{\mathsf{rcl}}, m_{\mathsf{rrsp}}, \mathsf{gpars})$: inputs a server S, a client message m_{rcl}, a response message m_{rrsp}, and attestation group parameters gpars, and outputs a bit $b \in \{0, 1\}$ to indicate whether the registration request was accepted. It also outputs the server-side context rc_S, a credential identifier cid, a session identifier sid, and agreed contents agCon from the perspective of the server S.

Authenticate: a two-pass challenge-response protocol run among a token T, a client C, and a server S after a successful run of Register, in which both T and S generated their registration contexts. At the end of Authenticate, S either accepts or rejects the authentication attempt. Similarly to Register, Authenticate can be decomposed into four algorithms:

- $m_{\mathsf{ach}} \xleftarrow{\$} \mathsf{aChal}(S, \mathsf{tb}, \mathsf{UV})$: inputs a server S, a token binding state tb, and a user verification condition $\mathsf{UV} \in \{\mathsf{T}, \mathsf{F}\}$, and outputs a challenge message m_{ach}. This algorithm does not change the state of the server S.
- $(m_{\mathsf{acl}}, m_{\mathsf{acom}}) \xleftarrow{\$} \mathsf{aCom}(\mathsf{id}_S, m_{\mathsf{ach}}, \mathsf{tb})$: run by the stateless client; it inputs the intended server identity id_S, a challenge message m_{ach}, and a token binding state tb, and outputs a client message m_{acl} and a command message m_{acom}.
- $(m_{\mathsf{arsp}}, \mathsf{rc}_T, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \xleftarrow{\$} \mathsf{aRsp}(T, m_{\mathsf{acom}})$: inputs a token T, and a command message m_{acom}, and outputs a response message m_{arsp}, the updated token-side registration context rc_T, a credential identifier cid, a session identifier sid, and agreed contents agCon from the perspective of the token T.
- $(b, \mathsf{rc}_S, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \xleftarrow{\$} \mathsf{aVrfy}(S, m_{\mathsf{acl}}, m_{\mathsf{arsp}})$: inputs a server S, a client message m_{acl}, and a response message m_{arsp}, and outputs a bit $b \in \{0, 1\}$ indicating whether the authentication request was accepted. It also outputs the updated server-side registration context rc_S, a credential identifier cid, a session identifier sid, and agreed contents agCon from the perspective of the server S.

Attestation modes. Unlike [7], our model captures only attestation modes None, Self, and Basic (also known as batch attestation), as the other modes attCA and anonCA are not as commonly used and, in particular, they are not used for USB tokens that rely on CTAP, the main focus of this work.

Therefore, for simplicity, we deviate from [7] and define a group initialization algorithm $(\mathsf{gpars}, \mathsf{rc}) \xleftarrow{\$} \mathsf{GInit}$ that creates a new group, which is cryptographically defined by some public group parameters gpars and a private registration context rc. This public gpars is taken as input by the servers and the private rc is taken as input by tokens in the same group as their initial registration context. For attestation modes None and Self, such attestation material gpars and rc are empty; while for the Basic mode, GInit uses a key generation algorithm to output an attestation key pair, then assign the private key to rc and assign the public key (with potentially other public parameters) to gpars.

[9]When we say an algorithm inputs a server, a client, or a token, we mean all of its state. The outputs of the algorithm make it explicit which parts of the state may be changed that are relevant for the security game.

G.2 PlA Authentication Model and WebAuthn Authentication

Again, we closely follow [6, 7] to define our authentication security model for PlA protocols. Our model is very similar to the ePlA model defined in [6] and the ePlAA model defined in [7], with some important changes that we discuss in this subsection below.

Trust model. For attestation modes None and Self, we assume that the PlA adversary is passive during registration, since the server has no prior knowledge of attestation material stored in the specific token of interest. Indeed, if the adversary can be active during registration, it is impossible to prove WebAuthn secure for these modes, as noted in [7]. For mode Basic, and as in [7], we can prove PlA security even with active registration, because the guarantee provided to the server is merely that the credential has been created by some token in the target group. However, the server does not know which token in the group produced the credential and, in fact, we know from our discussion of rogue key attacks and known unlinkability results that it is impossible for the server to know if this token is owned by a given user or by an adversary. As mentioned in the introduction, our fix to CTAP allows us to strengthen this authentication guarantee and assure the server that the token generating an attested credential is actually cryptographically bound to a specific client. We allow groups to be dynamically generated by the adversary. For attestation mode Basic, this corresponds to creating a new batch. The adversary is allowed to create as many groups as it wants and to assign tokens to these groups at will. (Note that in modes None and Self there is only one group.) Then, the adversary is allowed to corrupt the attestation material of all tokens except the tokens in the target batch of interest. Fine-grained credential corruption (not including the attestation material) is still allowed within the batch. This is a strengthening of the model in [7]; meanwhile it does not have a significant impact on the WebAuthn security proof and simplifies the description of the model.

Session oracles and registration contexts. To model concurrent or sequential PlA protocol instances (i.e., sessions) of a server S (associated with id_S) and sequential PlA sessions of a token T, we use $\pi^i_{r,S}$ and $\pi^j_{r,T}$ to denote their i-th and j-th registration instances, and $\pi^i_{a,S}$ and $\pi^j_{a,T}$ to denote their i-th and j-th authentication instances. The execution status of a session oracle $\pi^k_{ph,P}$ ($ph \in \{r, a\}$, $P \in \{S, T\}$), denoted by $\pi^k_{ph,P}.\mathsf{st}_{\mathsf{exe}}$, is either of $\{\perp, \mathsf{running}, \mathsf{accepted}\}$; here $\perp$ means the session oracle is not yet initialized, in which case we simply write $\pi^k_{ph,P} = \perp$. Session identifiers sid and agreed contents agCon are specific to a session. Registration contexts rc_S, rc_T are global to a server or token, respectively, and we abuse notation to allow them to be indexed by the (unique) identity of a token or server, respectively, as $\mathsf{rc}_S[T]$ or $\mathsf{rc}_T[S]$. This is well defined as we impose a single registration run between a given pair (S, T).

Session partnership. We say that a server registration session $\pi^i_{r,S}$ partners with a token registration session $\pi^j_{r,T}$ if and only if $\pi^i_{r,S}.\mathsf{sid} = \pi^j_{r,T}.\mathsf{sid}$. We define partnership for authentication sessions as $\pi^i_{a,S}.\mathsf{sid} = \pi^j_{a,T}.\mathsf{sid}$, and, furthermore, we require that they can be associated to unique partnered registration sessions $\pi^{i'}_{r,S}.\mathsf{sid} = \pi^{j'}_{r,T}.\mathsf{sid}$ such that $\pi^i_{a,S}.\mathsf{cid} = \pi^{i'}_{r,S}.\mathsf{cid}$. In other words, authentication partnership guarantees that there is unique registration partnership between the same server and token that establish the (unique) credential identifier that the server recovers at the end of the authentication run. Many authentication runs can, of course, be bound to the same registration sessions.

Looking ahead, with the above session partnership, our model is stronger than the models in [6, 7], in the sense that our model further guarantees that authentication binds the user to a unique registration that took place before between the same server/token pair, identified by a credential identifier cid. This also allows us to capture the typical scenario where cid is used by the server to identify the correct application-specific identifier. Note that our session partnership definition is, in the above sense, the same as [3].

Advantage measure. For a PlA protocol PlA, its advantage (with respect to the security experiment $\mathsf{Expt}^{\mathsf{pla\text{-}auth}}_{\mathsf{PlA}}$ shown in Figure 18) is defined for any adversary $\mathcal{A}$ as

$$\mathsf{Adv}^{\mathsf{pla\text{-}auth}}_{\mathsf{PlA}}(\mathcal{A}) = \Pr[\mathsf{Expt}^{\mathsf{pla\text{-}auth}}_{\mathsf{PlA}}(\mathcal{A}) = 1]$$

Authentication security of WebAuthn. We do not restate the authentication security of WebAuthn in this model, as it has been proved in [3]. Indeed, the security proofs given in [3] suffice to show that WebAuthn satisfies the PlA authentication notion we consider here: the proof begins by excluding collisions in credential identifiers, and then relies on the uniqueness of credential identifiers to pinpoint a unique registration session that established the public key under which the authentication took place. For attestation modes None, Self in which no hardcoded attestation material is used, the proof holds when the adversary is restricted only to passive registration attacks and can create only one group for which the attestation parameters are empty. For batch attestation the adversary can create an arbitrary number of groups. Rather than fixing a group a priori, we just allow the adversary to choose the target group adaptively, but this is of no consequence to the proof: as stated in [3], the only way the attacker could succeed in registering a key that is outside of the group fixed by a server verification run would be to either corrupt the group or forge an attestation signature.

Differences to prior work. As discussed above, our model introduces two differences to [6, 7], in order to capture a guarantee provided by FIDO2, which was captured by the original model in [3] but lost in the more recent works [6, 7]: a separation between registration and authentication sessions for modeling the partnership between them, and the explicit handling of credential identifiers and group identifiers.[10] As mentioned before, we simplify our model to capture the simplest attestation modes, as they are the most commonly used modes in practical USB-based tokens today and handling all possible attestation modes is not our focus. In particular, there is no interactive set-up of a token, which is required for the more complicated certification-based attestation modes. Instead, we tailor our definition to the settings where there is no attestation (or just simple Self attestation) or where batch attestation is used. For this, we let the server registration verification algorithm input some group parameters, and impose that the

[10]For simplicity, we also keep the algorithm negotiation parts of PlA implicit in the experiment code.

```
Partnerships$(S, i, \mathsf{cid})$:
1: if $\exists i'$ such that $\pi^{i'}_{r,S}.\mathsf{cid} = \mathsf{cid}$ then
2:     Retrieve $(S, i', \mathsf{gid}, \mathsf{cid})$ from $\mathcal{L}_{\mathsf{reg}}$
3:     if $\exists T, j'$ such that $\pi^{i'}_{r,S}.\mathsf{sid} = \pi^{j'}_{r,T}.\mathsf{sid}$ then
4:         if $\exists j$ such that $\pi^{i}_{a,S}.\mathsf{sid} = \pi^{j}_{a,T}.\mathsf{sid}$ then
5:             return $(\mathsf{gid}, i', T, j', j)$
6:         else return $(\mathsf{gid}, i', T, j', \perp)$
7:     else return $(\mathsf{gid}, i', \perp, \perp, \perp)$
8: return $(\perp, \perp, \perp, \perp, \perp)$
Win-auth $(S, i, \mathsf{cid})$:
1: if $\exists (S_1, i_1, ph_1) \neq (S_2, i_2, ph_2)$ s.t. $\pi^{i_1}_{ph_1,S_1}.\mathsf{sid} = \pi^{i_2}_{ph_2,S_2}.\mathsf{sid} \neq \perp$ then return 1
2: if $\exists (T_1, j_1, ph_1) \neq (T_2, j_2, ph_2)$ s.t. $\pi^{j_1}_{ph_1,T_1}.\mathsf{sid} = \pi^{j_2}_{ph_2,T_2}.\mathsf{sid} \neq \perp$ then return 1
3: if $\exists (S_1, i_1) \neq (S_2, i_2)$ s.t. $\pi^{i_1}_{r,S_1}.\mathsf{cid} = \pi^{i_2}_{r,S_2}.\mathsf{cid} \neq \perp$ then
4:     return 1
5: if $\exists (S', i', ph'), (T', j', ph')$ s.t. $\pi^{i'}_{ph',S'}.\mathsf{sid} = \pi^{j'}_{ph',T'}.\mathsf{sid} \neq \perp$ and $(S', T') \notin \mathcal{L}_{\mathsf{corr}}$ and
   $T'.\mathsf{gid} \notin \mathcal{L}_{\mathsf{corrG}}$ and $\pi^{i'}_{ph',S'}.\mathsf{agCon} \neq \pi^{j'}_{ph',T'}.\mathsf{agCon}$ then return 1
6: $(\mathsf{gid}, \_, T, \_, j) \leftarrow$ Partnerships$(S, i, \mathsf{cid})$
7: // Attestation broken: wrong group or no registration partner
8: if $\mathsf{gid} = \perp$ or $(\mathsf{gid} \notin \mathcal{L}_{\mathsf{corrG}}$ and $(T = \perp$ or $T.\mathsf{gid} \neq \mathsf{gid}))$ then
9:     return 1
10: else
11:     // Authentication broken: no authentication partner
12:     if $\mathsf{gid} \notin \mathcal{L}_{\mathsf{corrG}}$ then
13:         if $(S, T) \notin \mathcal{L}_{\mathsf{corr}}$ and $j = \perp$ then return 1
14:     return 0
```

```
Expt$^{\mathsf{pla\text{-}auth}}_{\mathsf{PlA}}(\mathcal{A})$:
1: $\mathcal{L}_{\mathsf{reg}} \leftarrow \emptyset$; $\mathcal{L}_{\mathsf{corr}} \leftarrow \emptyset$; $\mathcal{L}_{\mathsf{corrG}} \leftarrow \emptyset$; $G \leftarrow \emptyset$; $\mathsf{gid} \leftarrow 0$
2: $\mathsf{win\text{-}auth} \leftarrow 0$
3: $() \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(1^{\lambda})$
4: return win-auth
```

```
Reg $((S, i), (T, j), \mathsf{tb}, \mathsf{UV}, \mathsf{gid})$:
1: // This oracle replaces the rChall, rResp, rCompl oracles
   in the passive registration mode
2: if $T.\mathsf{gid} = \perp$ or $\pi^{i}_{r,S} \neq \perp$ or $\pi^{j}_{r,T} \neq \perp$ or $\mathsf{rc}_T[S] \neq \perp$ or
   $G[\mathsf{gid}] = \perp$ then return $\perp$
3: $m_{\mathsf{rch}} \xleftarrow{\$} \mathsf{rChal}(\pi^{i}_{r,S}, \mathsf{tb}, \mathsf{UV})$
4: $(m_{\mathsf{rcom}}, m_{\mathsf{rcl}}) \leftarrow \mathsf{rCom}(\mathsf{id}_S, m_{\mathsf{rch}}, \mathsf{tb})$
5: $(m_{\mathsf{rrsp}}, \mathsf{rc}_T, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \xleftarrow{\$} \mathsf{rRsp}(\pi^{j}_{r,T}, m_{\mathsf{rcom}})$
6: $(d, \mathsf{rc}_S, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \xleftarrow{\$}$
   $\mathsf{rVrfy}(\pi^{i}_{r,S}, m_{\mathsf{rcl}}, m_{\mathsf{rrsp}}, G[\mathsf{gid}].\mathsf{gpars})$
7: $\mathcal{L}_{\mathsf{reg}} \leftarrow \mathcal{L}_{\mathsf{reg}} \cup \{(S, i, \mathsf{gid}, \mathsf{cid})\}$
8: return $(d, m_{\mathsf{rch}}, m_{\mathsf{rcl}}, m_{\mathsf{rcom}}, m_{\mathsf{rrsp}})$
```

```
NewGroup ():
1: $(\mathsf{gpars}, \mathsf{rc}) \xleftarrow{\$} \mathsf{GInit}$
2: $G[\mathsf{gid}] \leftarrow (\mathsf{gpars}, \mathsf{rc})$
3: $\mathsf{gid} \leftarrow \mathsf{gid} + 1$
NewToken $(\mathsf{gid}, T)$:
1: if $G[\mathsf{gid}] = \perp$ then return $\perp$
2: if $T.\mathsf{gid} \neq \perp$ then return $\perp$
3: $T.\mathsf{gid} \leftarrow \mathsf{gid}$
4: $\mathsf{rc}_T \leftarrow G[\mathsf{gid}].\mathsf{rc}$
```

```
rChall $((S, i), \mathsf{tb}, \mathsf{UV})$:
1: if $\pi^{i}_{r,S} \neq \perp$ then return $\perp$
2: $m_{\mathsf{rch}} \xleftarrow{\$} \mathsf{rChal}(\pi^{i}_{r,S}, \mathsf{tb}, \mathsf{UV})$
3: return $m_{\mathsf{rch}}$
rCompl $((S, i), m_{\mathsf{rcl}}, m_{\mathsf{rrsp}}, \mathsf{gid})$:
1: if $\pi^{i}_{r,S} = \perp$ or $\pi^{i}_{r,S}.\mathsf{st}_{\mathsf{exe}} \neq \mathsf{running}$ or $G[\mathsf{gid}] = \perp$ then
   return $\perp$
2: $(d, \mathsf{rc}_S, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \xleftarrow{\$}$
   $\mathsf{rVrfy}(\pi^{i}_{r,S}, m_{\mathsf{rcl}}, m_{\mathsf{rrsp}}, G[\mathsf{gid}].\mathsf{gpars})$
3: if $d = 1$ then $\mathcal{L}_{\mathsf{reg}} \leftarrow \mathcal{L}_{\mathsf{reg}} \cup \{(S, i, \mathsf{gid}, \mathsf{cid})\}$
4: return $d$
```

```
Corrupt $(S, T)$:
1: if $\mathsf{rc}_T[S] = \perp$ then return $\perp$
2: $\mathcal{L}_{\mathsf{corr}} \leftarrow \mathcal{L}_{\mathsf{corr}} \cup \{(S, T)\}$
3: return $\mathsf{rc}_T[S]$

rResp $((T, j), m_{\mathsf{rcom}})$:
1: if $\pi^{j}_{r,T} \neq \perp$ or $T.\mathsf{gid} = \perp$ then
   return $\perp$
2: $(m_{\mathsf{rrsp}}, \mathsf{rc}_T, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \xleftarrow{\$}$
   $\mathsf{rRsp}(\pi^{j}_{r,T}, m_{\mathsf{rcom}})$
3: $\mathcal{L}^{\mathsf{r}}_{\mathsf{ch}} \leftarrow \mathcal{L}^{\mathsf{r}}_{\mathsf{ch}} \cup \{\mathsf{cid}\}$
4: return $m_{\mathsf{rrsp}}$
```

```
aChall $((S, i), \mathsf{tb}, \mathsf{UV})$:
1: if $\pi^{i}_{a,S} \neq \perp$ then return $\perp$
2: $m_{\mathsf{ach}} \xleftarrow{\$} \mathsf{aChal}(\pi^{i}_{a,S}, \mathsf{tb}, \mathsf{UV})$
3: return $m_{\mathsf{ach}}$
aCompl $((S, i), m_{\mathsf{acl}}, m_{\mathsf{arsp}})$:
1: if $\pi^{i}_{a,S} = \perp$ or $\pi^{i}_{a,S}.\mathsf{st}_{\mathsf{exe}} \neq \mathsf{running}$ then return $\perp$
2: $(d, \mathsf{rc}_S, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \xleftarrow{\$} \mathsf{aVrfy}(\pi^{i}_{a,S}, m_{\mathsf{acl}}, m_{\mathsf{arsp}})$
3: if $d = 1$ and win-auth $= 0$ then
4:     win-auth $\leftarrow$ Win-auth$(S, i, \mathsf{cid})$
5: return $d$
```

```
CorruptGroup $(\mathsf{gid})$:
1: if $G[\mathsf{gid}] = \perp$ then return $\perp$
2: $\mathcal{L}_{\mathsf{corrG}} \leftarrow \mathcal{L}_{\mathsf{corrG}} \cup \{\mathsf{gid}\}$
3: return $G[\mathsf{gid}]$
aResp $((T, j), m_{\mathsf{acom}})$:
1: if $\pi^{j}_{a,T} \neq \perp$ or $T.\mathsf{gid} = \perp$ then
   return $\perp$
2: $(m_{\mathsf{arsp}}, \mathsf{rc}_T, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \xleftarrow{\$}$
   $\mathsf{aRsp}(\pi^{j}_{a,T}, m_{\mathsf{acom}})$
3: $\mathcal{L}^{\mathsf{a}}_{\mathsf{ch}} \leftarrow \mathcal{L}^{\mathsf{a}}_{\mathsf{ch}} \cup \{\mathsf{cid}\}$
4: return $m_{\mathsf{arsp}}$
```

Figure 18: Security experiment, winning conditions, and oracle definitions for PlA authentication security experiment. Code in blue represents the added modifications with respect to [6, 7]. Code in teal is unique to the PlA privacy experiment as defined in Figure 19. We let $\mathcal{O}$ denote the set of all of the security experiment oracles that are available to $\mathcal{A}$. The winning condition procedure Win-auth is called in the aCompl oracle whenever the server authentication session accepts.

server can only accept registrations from tokens whose attestation material is consistent with a given set of group parameters.

G.3 PlA Privacy Model and WebAuthn Privacy

Our privacy PlA model closely follows [7]. Except for the changes in PlA oracles and picking the attestation group that are already explained in Section G.2, we change how context separation is checked. Instead of adding token and indices to $\mathcal{L}$, we decide to follow [17] and add credential id cid to $\mathcal{L}$ instead. Compared to [7] adding token itself to $\mathcal{L}$, this has two advantages: 1) We allow tokens to be registered and authenticated multiple times in Phase 1 and Phase 3, and we only prohibit authentications on the **particular** registration request that is done via LEFT/RIGHT oracles. [7] prohibits all regular authentication requests, if the challenge token is ever registered via LEFT/RIGHT oracles. 2) This prevents trivial attack caused by index colliding. Consider the scenario: an attacker can perform Register-LEFT, then perform a regular authenticate on token T_0 (suppose two challenge tokens are T_0 and T_1). In [7], this will be allowed. However, the attacker can identify which token is used by Register-LEFT. If Register-LEFT uses T_0, the authentication will succeed (although the attacker loses eventually), if Register-LEFT uses T_1, the authentication will fail, and the attacker thus conclude the bit b is 1.

Additionally, [17] requires instance freshness, which prohibits the attacker from querying regular oracles on the j-th instace of challenge token if the j-th instance of **corresponding** token is used in LEFT/RIGHT oracles. We strengthen that requirement to prohibit the attacker from querying regular oracle on j-th instance of **either** challenge tokens, if the j-th instance of at least one token is used inside LEFT/RIGHT oracle. Consider the following scenario: an attacker can query Register-LEFT, then query regular Register(T_0, 0). If Register-LEFT uses T_0, the registration will fail (and the attacker loses eventually), if Register-LEFT uses T_1, the registration will succeed, and the attacker thus conclude the bit b is 1. We stress that the above two attacks are not real-world attacks, and do not refute privacy results in [7] [17]. They are merely "model attacks" caused by collision of indices. Nonetheless, we fix them in our model.

We provide the privacy experiment in detail in Figure 19. For any adversary $\mathcal{A}$, its advantage is defined as

$$\mathsf{Adv}^{\mathsf{priv}}_{\mathsf{PlA}}(\mathcal{A}) = |2\Pr[\mathsf{Expt}^{\mathsf{priv}}_{\mathsf{PlA}}(\mathcal{A}) = 1] - 1| \,.$$

We remark that in [7] the advantage is incorrectly defined as the probability of the experiment returning 1. With such definition, an adversary randomly guessing the challenge bit will have advantage 1/2, which is non-negligible. In [17] the advantage is defined as how we define it above, but the experiment returns 0 if the adversary does not follow the rules. But then the advantage of such adversary will be 1. In our definition, the experiment returns a random bit in case the adversary misbehaves, yielding advantage 0, as expected.

We then establish the following theorem:

Theorem 7. *For any adversary $\mathcal{A}$, we have that*

$$Adv^{priv}_{WebAuthn}(\mathcal{A}) = 0$$

We note that the proofs in [7] still work for the stronger model.

```
$\mathsf{Expt}^{\mathsf{priv}}_{\mathsf{PlA}}(\mathcal{A})$:
  1: $\mathcal{L}_{\mathsf{corr}} \leftarrow \emptyset, \mathcal{L}^{\mathsf{r}}_{\mathsf{ch}} \leftarrow \emptyset, \mathcal{L}^{\mathsf{a}}_{\mathsf{lr}} \leftarrow \emptyset, \mathcal{L}^{\mathsf{a}}_{\mathsf{ch}} \leftarrow \emptyset, \mathcal{L}^{\mathsf{r}}_{\mathsf{lr}} \leftarrow \emptyset$
  2: $st_1 \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(1^\lambda)$ // Phase 1
  3: $T_0, T_1, S_L, S_R, st_2 \xleftarrow{\$} \mathcal{A}(1^\lambda, st_1)$ // Phase 2
  4: $b \leftarrow \mathrm{InitRL}(T_0, T_1, S_L, S_R)$
  5: $\mathcal{O}' \leftarrow (\mathcal{O} \backslash \{\textbf{NewToken}\})$
  6: $b' \leftarrow \mathcal{A}^{\mathcal{O}',\mathsf{LEFT},\mathsf{RIGHT}}(1^\lambda, st_2)$ // Phase 3
  7: $r \xleftarrow{\$} \{0,1\}$
  8: if Check-priv-PlA() then return $b = b'$
  9: else return $r$

Check-priv-PlA($b, b'$):
  1: $\mathsf{S} \leftarrow (\mathcal{L}^{\mathsf{r}}_{\mathsf{ch}} \cap \mathcal{L}^{\mathsf{a}}_{\mathsf{lr}}) \cup (\mathcal{L}^{\mathsf{a}}_{\mathsf{ch}} \cap \mathcal{L}^{\mathsf{r}}_{\mathsf{lr}}) \cup (\mathcal{L}^{\mathsf{r}}_{\mathsf{ch}} \cap \mathcal{L}^{\mathsf{r}}_{\mathsf{lr}})$
  2: if $b = b'$ and $\mathsf{S} = \emptyset$ and $(S_L, T_0), (S_R, T_1), (S_L, T_1), (S_R, T_0) \notin \mathcal{L}_{\mathsf{corr}}$ and $T_0.\mathsf{gid}{=}T_1.\mathsf{gid}$ then
  3:     return 1
  4: else
  5:     return 0

InitRL($T_0, T_1, S_L, S_R$):
  1: $b \xleftarrow{\$} \{0,1\}$
  2: Initialize oracles r/aLEFT$_{T_b,S_L}$ and r/aRIGHT$_{T_{1-b},S_R}$
  3: return $b$

r/aLEFT$_{T_b,S_L}(m)$
  1: Obtains intended server S from m
  2: if $S \neq S_L$ then
  3:     return $\bot$
  4: $j \leftarrow 0$ while $\pi^j_{T_b} \neq \bot$:
  5: $j \leftarrow j+1$
  6: return rResp' $((T_b, j), m)$ // in $r$LEFT
  7: return aResp' $((T_b, j), m)$ // in $a$LEFT

r/aRIGHT$_{T_{1-b},S_R}(m)$
  1: Obtains intended server S from m
  2: if $S \neq S_R$ then
  3:     return $\bot$
  4: $j \leftarrow 0$ while $\pi^j_{T_{1-b}} \neq \bot$:
  5: $j \leftarrow j+1$
  6: return rResp' $((T_{1-b}, j), m)$ // in $r$RIGHT
  7: return aResp' $((T_{1-b}, j), m)$ // in $a$RIGHT

rResp' $((T, j), m_{\mathsf{acom}})$: // helper function
  1: if $\pi^j_T \neq \bot$ or $T.\mathsf{gid} = \bot$ then
  2:     return $\bot$
  3: $(m_{\mathsf{rrsp}}, \mathsf{rc}_T, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \xleftarrow{\$} \mathsf{rResp}((T, j), m_{\mathsf{acom}})$
  4: $\mathcal{L}^{\mathsf{r}}_{\mathsf{lr}} \xleftarrow{\cup} \{\mathsf{cid}\}$
  5: return $m_{\mathsf{rrsp}}$

aResp' $((T, j), m_{\mathsf{acom}})$: // helper function
  1: if $\pi^j_T \neq \bot$ or $T.\mathsf{gid} = \bot$ then
  2:     return $\bot$
  3: $(m_{\mathsf{arsp}}, \mathsf{rc}_T, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \xleftarrow{\$} \mathsf{aResp}((T, j), m_{\mathsf{acom}})$
  4: $\mathcal{L}^{\mathsf{a}}_{\mathsf{lr}} \xleftarrow{\cup} \{\mathsf{cid}\}$
  5: return $m_{\mathsf{rrsp}}$
```

Figure 19: Experiment $\mathsf{Expt}^{\mathsf{priv}}_{\mathsf{PlA}}$ for PlA privacy with oracles $\mathcal{O}$ defined in Fig.18. Similar to Fig.18 , code in blue represents the added modifications with respect to [7].

H Composed Authentication Model

We introduce our composed model for authentication security, based on the PlA authentication security model presented in Appendix G and the proposed mPACA model from section 4.

Following the approach from [3, 6], we consider a security experiment $\mathsf{Expt}^{\mathsf{ua}}_{\mathsf{PlA+mPACA}}$, presented in figures 21, 20, which is executed between a challenger and an adversary $\mathcal{A}$ against the ua (*user authentication*) security of PlA+mPACA.

Trust model. Like in [3, 6], we assume the communication channel between servers and clients is authenticated in the sense that the client is assured as to the identity of the server (capturing the guarantees of a secure connection, e.g., established by TLS). We maintain the trust model from mPACA unchanged, which means any client session can only complete Bind passively. However, we now allow active composed model adversaries during registration runs, regardless of the attestation mode, whereas in the PlA model we can only deal with active registration when using Basic (batch) attestation. Indeed the only difference between attestation modes None/Self with respect to mode Basic is that for the former we do not allow mPACA clients to be compromised during registration, whereas for the latter we can allow this. We note that this is a stronger model than that adopted in [6], in that we can deal with active attacks during registration, because we have upgraded PACA to mPACA to provide a bidirectional authenticated channel.

Session oracles and partnership. We maintain the session oracles defined for PlA and mPACA, as well as all protocol variables, internal states and partnership definitions, but follow the approach from [6] by using $\tilde{\pi}$ and π to refer to PlA and mPACA sessions respectively.

Experiment oracles. The ua experiment gives the adversary access to all of the SUF-t experiment's oracles except for Auth-C, Validate-T, Auth-T and Validate-C. Furthermore, the adversary also has access to all unchanged oracles from the pla-auth experiment, except for rChall, rResp, rCompl, aChall, aResp and aCompl, which are redefined for this experiment in Figure 20, and except for Reg, which is absent due to the assumption that $\mathcal{A}$ can always actively interfere with any registration session. The rChall oracle now additionally takes a client session as input, which prepares the message from the server session $\tilde{\pi}^i_{r,S}$ by calling rCom to produce a client message m_{rcl} and a command m_{rcom}. This command is then authenticated using the mPACA oracle Auth-C, producing a tag t_{cl}. In addition to m_{rch}, both messages output by rCom and t_{cl} are given to $\mathcal{A}$. The changes to aChall are analogous. The rResp oracle now additionally inputs an mPACA token session, a tag t_{cl} and a user decision bit d. It starts by querying the Validate-T oracle on the m_{rcom} message and tag t_{cl}, aborting if the status is not accepted. Then, it authenticates the PlA token response m_{rrsp} via the mPACA oracle Auth-T, producing a tag t_{tk}, which is also given to $\mathcal{A}$. The changes to aResp are analogous. Finally, the rCompl oracle additionally receives a tag t_{tk}, fetches the client session associated with the current server session, and queries Validate-C on the token response m_{rrsp} and t_{tk}, failing if the status is not accepted. The changes to aCompl are analogous, with the addition that it now calls Win-ua (described below) to verify the winning conditions whenever $\tilde{\pi}^i_{a,S}$ accepts. We also define two lists $\mathcal{L}_{\mathsf{pla\text{-}paca\text{-}S}}$ and $\mathcal{L}_{\mathsf{pla\text{-}paca\text{-}T}}$ to, respectively, link every PlA server session with its associated mPACA client session in rChall and aChall, and every PlA token session with its associated mPACA token session in rResp and aResp.

Winning conditions and advantage measure. The winning conditions specified in Win-ua (see Figure 21) intuitively guarantee that PlA and mPACA sessions can be uniquely identified by their derived session identifiers and, furthermore, as in the PlA experiments, that registration sessions obtain unique credential identifiers. Then, the adversary wins if it breaks the PlA authentication security at any point, regardless of whether it is using compromised mPACA clients or not. Furthermore, when using uncompromised mPACA clients, the adversary also wins if it can convince the PlA server to pair with a registration or authentication session that is hosted by some other token than the one bound to the unique client that it communicates with. For a composed PlA+mPACA protocol, its ua advantage is defined for any adversary $\mathcal{A}$ as

$$\mathsf{Adv}^{\mathsf{ua}}_{\mathsf{PlA+mPACA}}(\mathcal{A}) = \Pr[\mathsf{Expt}^{\mathsf{ua}}_{\mathsf{PlA+mPACA}}(\mathcal{A}) = 1]$$

Differences to prior work. We adopt the style of presentation of [6], but our composed model expresses the winning condition in a way that is closer to the one defined in [3]: we express the adversary's advantage purely as a function of its probability of breaking a server-side authentication guarantee and we do not include PACA-specific command forgery checks.[11] We can do this because we establish a stronger result based on mPACA. A winning condition closer to the one in [6] needs to be considered to capture the composed security of the current version current of FIDO2, as discussed at the end of Appendix I. Furthermore, our

[11]The authors in [3] also consider user-side guarantees, which are also provided by FIDO2, but we do not consider them here and leave clarifying and expanding the study of these guarantees as future work.

```
rChall $(S, i, C, k, \text{tb}, UV)$:
  1: if $\tilde{\pi}^{i}_{r,S} \neq \perp$ then return $\perp$
  2: $m_{\text{rch}} \xleftarrow{\$} \text{rChal}(\tilde{\pi}^{i}_{r,S}, \text{tb}, \text{UV})$
  3: $(m_{\text{rcl}}, m_{\text{rcom}}) \xleftarrow{\$} \text{rCom}(\text{id}_S, m_{\text{rch}}, \text{tb})$
  4: $\text{resp} \xleftarrow{\$} \text{Auth-C}(C, k, m_{\text{rcom}})$
  5: if $\text{resp} = \perp$ then return $\perp$
  6: $(m_{\text{rcom}}, t_{\text{cl}}) \leftarrow \text{resp}$
  7: $\mathcal{L}_{\text{pla-paca-S}} \leftarrow \mathcal{L}_{\text{pla-paca-S}} \cup \{(\text{reg}, S, i, C, k)\}$
  8: return $(m_{\text{rch}}, m_{\text{rcl}}, m_{\text{rcom}}, t_{\text{cl}})$
rResp $(T, j, j', m_{\text{rcom}}, t_{\text{cl}}, d)$:
  1: if $\tilde{\pi}^{j}_{r,T} \neq \perp$ or $T.\text{gid} = \perp$ then return $\perp$
  2: $\text{status} \xleftarrow{\$} \text{Validate-T}(T, j', m_{\text{rcom}}, t_{\text{cl}}, d)$
  3: if $\text{status} \neq \text{accepted}$ then return $\perp$
  4: $(m_{\text{rrsp}}, \text{rc}_T, \text{cid}, \text{sid}, \text{agCon}) \xleftarrow{\$} \text{rRsp}(\tilde{\pi}^{j}_{r,T}, m_{\text{rcom}})$
  5: $(m_{\text{rrsp}}, t_{\text{tk}}) \xleftarrow{\$} \text{Auth-T}(T, j', m_{\text{rrsp}})$
  6: $\mathcal{L}_{\text{pla-paca-T}} \leftarrow \mathcal{L}_{\text{pla-paca-T}} \cup \{(\text{reg}, T, j, j')\}$
  7: return $(m_{\text{rrsp}}, t_{\text{tk}})$
rCompl $(S, i, m_{\text{rcl}}, m_{\text{rrsp}}, \text{gid}, t_{\text{tk}})$:
  1: if $\tilde{\pi}^{i}_{r,S} = \perp$ or $\tilde{\pi}^{i}_{r,S}.\text{st}_{\text{exe}} \neq \text{running}$ or $G[\text{gid}] = \perp$ then return $\perp$
  2: Retrieve $(\text{reg}, S, i, C, k)$ from $\mathcal{L}_{\text{pla-paca-S}}$
  3: $\text{status} \leftarrow \text{Validate-C}(C, k, m_{\text{rrsp}}, t_{\text{tk}})$
  4: if $\text{status} \neq \text{accepted}$ then return $\perp$
  5: $(d, \text{rc}_S, \text{cid}, \text{sid}, \text{agCon}) \xleftarrow{\$} \text{rVrfy}(\tilde{\pi}^{i}_{r,S}, m_{\text{rcl}}, m_{\text{rrsp}}, G[\text{gid}].\text{gpars})$
  6: if $d = 1$ then
  7:     $\mathcal{L}_{\text{reg}} \leftarrow \mathcal{L}_{\text{reg}} \cup \{(S, i, \text{gid}, \text{cid})\}$
  8: return $d$
```

```
aChall $(S, i, C, k, \text{tb}, \text{UV})$:
  1: if $\tilde{\pi}^{i}_{a,S} \neq \perp$ then return $\perp$
  2: $m_{\text{ach}} \xleftarrow{\$} \text{aChal}(\tilde{\pi}^{i}_{a,S}, \text{tb}, \text{UV})$
  3: $(m_{\text{acl}}, m_{\text{acom}}) \xleftarrow{\$} \text{aCom}(\text{id}_S, m_{\text{ach}}, \text{tb})$
  4: $\text{resp} \xleftarrow{\$} \text{Auth-C}(C, k, m_{\text{acom}})$
  5: if $\text{resp} = \perp$ then return $\perp$
  6: $(m_{\text{acom}}, t_{\text{cl}}) \leftarrow \text{resp}$
  7: $\mathcal{L}_{\text{pla-paca-S}} \leftarrow \mathcal{L}_{\text{pla-paca-S}} \cup \{(\text{auth}, S, i, C, k)\}$
  8: return $(m_{\text{ach}}, m_{\text{acl}}, m_{\text{acom}}, t_{\text{cl}})$
aResp $(T, j, j', m_{\text{acom}}, t_{\text{cl}}, d)$:
  1: if $\tilde{\pi}^{j}_{a,T} \neq \perp$ or $T.\text{gid} = \perp$ then return $\perp$
  2: $\text{status} \xleftarrow{\$} \text{Validate-T}(T, j', m_{\text{acom}}, t_{\text{cl}}, d)$
  3: if $\text{status} \neq \text{accepted}$ then return $\perp$
  4: $(m_{\text{arsp}}, \text{rc}_T, \text{cid}, \text{sid}, \text{agCon}) \xleftarrow{\$} \text{aRsp}(\tilde{\pi}^{j}_{a,T}, m_{\text{acom}})$
  5: $(m_{\text{arsp}}, t_{\text{tk}}) \xleftarrow{\$} \text{Auth-T}(T, j', m_{\text{arsp}})$
  6: $\mathcal{L}_{\text{pla-paca-T}} \leftarrow \mathcal{L}_{\text{pla-paca-T}} \cup \{(\text{auth}, T, j, j')\}$
  7: return $(m_{\text{arsp}}, t_{\text{tk}})$
aCompl $(S, i, m_{\text{acl}}, m_{\text{arsp}}, t_{\text{tk}})$:
  1: if $\tilde{\pi}^{i}_{a,S} = \perp$ or $\tilde{\pi}^{i}_{a,S}.\text{st}_{\text{exe}} \neq \text{running}$ then return $\perp$
  2: Retrieve $(\text{auth}, S, i, C, k)$ from $\mathcal{L}_{\text{pla-paca-S}}$
  3: $\text{status} \leftarrow \text{Validate-C}(C, k, m_{\text{arsp}}, t_{\text{tk}})$
  4: if $\text{status} \neq \text{accepted}$ then return $\perp$
  5: $(d, \text{rc}_S, \text{cid}, \text{sid}, \text{agCon}) \xleftarrow{\$} \text{aVrfy}(\tilde{\pi}^{i}_{a,S}, m_{\text{acl}}, m_{\text{arsp}})$
  6: if $d = 1$ and $\text{win-ua} = 0$ then
  7:     $\text{win-ua} \leftarrow \text{Win-ua}(S, i, \text{cid})$
  8: return $d$
```

Figure 20: Oracle definitions for ua security experiment for the composed model. Code in blue highlights the differences to the PlA oracles shown in G. Differences that are specific from mPACA are colored in red.

composed model inherits from our mPACA and PlA definitions the strengthenings we introduced for each of these primitives, namely the ability to dynamically choose attestation groups and the explicit guarantee that a PlA authentication session is bound to a unique registration session in the same token. Finally, the main novelty is that we explicitly deal with active attacks in the composed model for the registration phase, even for attestation modes None and Self, even though in these settings we need to restrict ourselves to uncompromised clients in registration.

I Authentication Security of FIDO2

Here we first present our main result for composed security relying on mPACA and later explain the weaker guarantees provided by current FIDO2.

Composed security of PlA+mPACA. The following theorem shows that mPACA guarantees a strong form of composed security than in prior works, since we can allow partial active attacks (the realistic model when using USB tokens as discussed in the introduction) during both registration and for all modes even without attestation. The intuition here is that the composed model speaks only about the guarantees provided to servers that are uniquely bound to honest clients that are out of the adversary's control: excluding a break of mPACA, we know that such clients (if uncompromised) will be communicating with a unique token instance via a bi-directional secure channel, which means that we do not

```
$\text{Expt}^{\text{ua}}_{\text{PlA+mPACA}}(\mathcal{A})$:
  1: $\mathcal{L}_{\text{reg}} \leftarrow \emptyset$; $\mathcal{L}_{\text{corr}} \leftarrow \emptyset$; $\mathcal{L}_{\text{corrG}} \leftarrow \emptyset$; $G \leftarrow \emptyset$; $\text{gid} \leftarrow 0$ // From PlA
  2: $\mathcal{L}_{\text{authC}}, \mathcal{L}_{\text{authT}} \leftarrow \emptyset$ // From mPACA
  3: $\mathcal{L}_{\text{pla-paca-S}}, \mathcal{L}_{\text{pla-paca-T}} \leftarrow \emptyset$ // For grouping PlA and mPACA sessions
  4: $\text{win-ua} \leftarrow 0$
  5: $() \xleftarrow{\$} \mathcal{A}^{O}(1^{\lambda})$
  6: return win-ua
Win-ua $(S, i, \text{cid})$:
  1: // If there exists any collision between server or token PlA sessions, $\mathcal{A}$ wins
  2: if $\exists (S_1, i_1, ph_1) \neq (S_2, i_2, ph_2)$ s.t. $\tilde{\pi}^{i_1}_{ph_1,S_1}.\text{sid} = \tilde{\pi}^{i_2}_{ph_2,S_2}.\text{sid} \neq \perp$ then return 1
  3: if $\exists (T_1, j_1, ph_1) \neq (T_2, j_2, ph_2)$ s.t. $\tilde{\pi}^{j_1}_{ph_1,T_1}.\text{sid} = \tilde{\pi}^{j_2}_{ph_2,T_2}.\text{sid} \neq \perp$ then return 1
  4:
  5: // If there exists any collision between client or token mPACA sessions, $\mathcal{A}$ wins
  6: if $\exists (C_1, k_1), (C_2, k_2)$ $s.t.$ $(C_1, j_1) \neq (C_2, j_2)$ and $\pi^{k_1}_{C_1}.\text{st}_{\text{exe}} = \pi^{k_2}_{C_2}.\text{st}_{\text{exe}} = \text{bindDone}$
       and $\pi^{k_1}_{C_1}.\text{sid} = \pi^{k_2}_{C_2}.\text{sid}$ then return 1
  7: if $\exists (T_1, j'_1), (T_2, j'_2)$ $s.t.$ $(T_1, j'_1) \neq (T_2, j'_2)$ and $\pi^{j'_1}_{T_1}.\text{st}_{\text{exe}} = \pi^{j'_2}_{T_2}.\text{st}_{\text{exe}} = \text{bindDone}$
       and $\pi^{j'_1}_{T_1}.\text{sid} = \pi^{j'_2}_{T_2}.\text{sid}$ then return 1
  8:
  9: // If there exist two distinct server sessions with the same cid, $\mathcal{A}$ wins
 10: if $\exists (S_1, i_1) \neq (S_2, i_2)$ s.t. $\tilde{\pi}^{i_1}_{r,S_1}.\text{cid} = \tilde{\pi}^{i_2}_{r,S_2}.\text{cid} \neq \perp$ then return 1
 11:
 12: // If there exists a server session and a token session that agree on the sid but not on the agCon, $\mathcal{A}$ wins.
     This captures a rogue key attack by registering a key from another batch.
 13: if $\exists (S', i', ph'), (T', j', ph')$ s.t. $\tilde{\pi}^{i'}_{ph',S'}.\text{sid} = \tilde{\pi}^{j'}_{ph',T'}.\text{sid} \neq \perp$ and $(S', T') \notin \mathcal{L}_{\text{corr}}$ and
       $T'.\text{gid} \notin \mathcal{L}_{\text{corrG}}$ and $\tilde{\pi}^{i'}_{ph',S'}.\text{agCon} \neq \tilde{\pi}^{j'}_{ph',T'}.\text{agCon}$ then return 1
 14:
 15: // If the server session $\tilde{\pi}^{i}_{a,S}$ that accepted has no corresponding server registration session, $\mathcal{A}$ wins. If $\mathcal{A}$
     never corrupted the token group associated with $\tilde{\pi}^{i'}_{r,S}$, and $\tilde{\pi}^{i}_{a,S}$ has no registration partner $T$, or $\tilde{\pi}^{i}_{a,S}$
     registered with the wrong gid, $\mathcal{A}$ wins. If $\tilde{\pi}^{i}_{a,S}$ is not partnered with one of $T$'s authentication sessions,
     and $\mathcal{A}$ never corrupted $T$, $\mathcal{A}$ wins
 16: $(\text{gid}, i', T, j', j) \leftarrow \text{Partnerships}(S, i, \text{cid})$
 17: if $\text{gid} = \perp$ or ($\text{gid} \notin \mathcal{L}_{\text{corrG}}$ and ($T = \perp$ or $T.\text{gid} \neq \text{gid}$)) then return 1
 18: else
 19:     if $\text{gid} \notin \mathcal{L}_{\text{corrG}}$ then
 20:         // The PlA registration token session $\tilde{\pi}^{j'}_{r,T}$ that is partnered with $\tilde{\pi}^{i'}_{r,S}$ is associated with an mPACA
             token session $\pi^{l'}_{T}$ that is the Bind partner of a client session $\pi^{k_1}_{C_1}$ that is partnered with $\tilde{\pi}^{i'}_{r,S}$
             (unless $\pi^{k_1}_{C_1}$ was compromised).
 21:         Retrieve $(\text{reg}, T, j', l')$ from $\mathcal{L}_{\text{pla-paca-T}}$ // Because we know at this point that $\tilde{\pi}^{j'}_{r,T}$ has completed
             a registration with $\tilde{\pi}^{i'}_{r,S}$ (from Partnerships), we know from the code in rResp that mPACA session
             $\pi^{l'}_{T}$ must exist in the list
 22:         Retrieve $(\text{reg}, S, i', C_S, k_S)$ from $\mathcal{L}_{\text{pla-paca-S}}$
 23:         $(C_T, k_T) \leftarrow \text{tokenBindPartner}(T, l')$
 24:         // The client $C_T$ to which the token $T$ is bound is *not* the same as $C_S$, which is the client linked to
             the server session $\tilde{\pi}^{i}_{S}$
 25:         if $(C_S, k_S) \neq (C_T, k_T)$ then
 26:             $(T_S, m) \leftarrow \text{clientBindPartner}(C_S, k_S)$
 27:             if $\pi^{k_S}_{C_S}.\text{compromised} = \text{false}$ and $\pi^{m}_{T_S}.\text{pinCorr} = \text{false}$ then return 1
 28:             if $((C_T, k_T) = (\perp, \perp)$ or $\pi^{k_T}_{C_T}.\text{compromised} = \text{false})$ and $\pi^{l'}_{T}.\text{pinCorr} = \text{false}$ then return
                 1
 29:         if $(S, T) \notin \mathcal{L}_{\text{corr}}$ and $j = \perp$ then return 1
 30:         else if $(S, T) \notin \mathcal{L}_{\text{corr}}$ then
 31:             // The PlA authentication token session $\tilde{\pi}^{j}_{a,T}$ that is partnered with $\tilde{\pi}^{i}_{a,S}$ is associated with an
                 mPACA token session $\pi^{l}_{T}$ that is the Bind partner of a client session $\pi^{k_2}_{C_2}$ that is partnered
                 with $\tilde{\pi}^{i}_{a,S}$ (unless $\pi^{k_2}_{C_2}$ was compromised).
 32:             Retrieve $(\text{auth}, T, j, l)$ from $\mathcal{L}_{\text{pla-paca-T}}$ // Because we know at this point that $\tilde{\pi}^{j}_{a,T}$ has
                 completed an authentication with $\tilde{\pi}^{i}_{a,S}$, we know from the code in aResp that mPACA session
                 $\pi^{l}_{T}$ must exist in the list
 33:             Retrieve $(\text{auth}, S, i, C'_S, k'_S)$ from $\mathcal{L}_{\text{pla-paca-S}}$
 34:             $(C'_T, k'_T) \leftarrow \text{tokenBindPartner}(T, l)$
 35:             if $(C'_S, k'_S) \neq (C'_T, k'_T)$ then
 36:                 $(T'_S, m') \leftarrow \text{clientBindPartner}(C_S, k_S)$
 37:                 if $\pi^{k'_S}_{C'_S}.\text{compromised} = \text{false}$ and $\pi^{m'}_{T'_S}.\text{pinCorr} = \text{false}$ then return 1
 38:                 if $((C'_T, k'_T) = (\perp, \perp)$ or $\pi^{k'_T}_{C'_T}.\text{compromised} = \text{false})$ and $\pi^{l}_{T}.\text{pinCorr} = \text{false}$ then
                     return 1
 39: return 0
```

Figure 21: The ua security experiment and winning conditions for the PlA+mPACA composed model. We call O the set of all of the security experiment's oracles that are available to $\mathcal{A}$. The winning condition procedure Win-ua is called in the aCompl oracle whenever the server session accepts. Composed model-specific winning conditions are in blue.

need to consider an active PlA attacker in the analysis of composed model security.

We provide two theorems, which intuitively capture the security guarantees for different attestation modes.

Theorem 8. *(Attestation* none *and* self*) Consider the setting in which the adversary is only allowed to create a single attestation group and forbid the possibility to compromise clients involved in registration sessions.*

Then, if there exists an adversary $\mathcal{A}$ against the composed security of an mPACA protocol mPACA and a PlA protocol PlA, then there exist adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ such that:

$$\mathsf{Adv}^{ua}_{PlA+mPACA}(\mathcal{A}) \leq \mathsf{Adv}^{SUF\text{-}t}_{mPACA}(\mathcal{B}_1) + \mathsf{Adv}^{pla\text{-}auth}_{PlA}(\mathcal{B}_2) \; .$$

Furthermore, $\mathcal{B}_2$ only requires access to a passive PlA registration oracle.

Proof. *(Sketch.)* Our proof proceeds in two game hops. In the first hop, we modify the composed security model to declare the adversary a loser whenever it breaks the mPACA security guarantee when interacting with oracles **rResp**, **rCompl**, **aResp** or **aCompl** (seen in Fig. 22). Any adversary $\mathcal{A}$ for which the probability of winning the composed security game varies visibly can be transformed into an mPACA attacker $\mathcal{B}_1$ with the same advantage via a trivial reduction. Note that $\mathcal{B}_1$ controls all the details of the PlA protocol in this reduction. Also observe that, after this hop, and because we disallow the compromise of clients involved in registration runs, the adversary is now restricted to passive behavior when dealing with these sessions.

In the second hop, we declare the adversary a loser if it breaks a PlA guarantee (seen in Fig. 23). We reduce any distinguishing advantage between the two games by constructing a reduction $\mathcal{B}_2$ to PlA security. Note here that we need to argue that we can program the trace of a passive registration run into the composed model oracles, where the adversary has some active attacking power. Here is how the reduction can do this:

- when the adversary calls **rChall**, there are two cases. Either the client has a unique token partner, or it does not have any binding state: this is guaranteed by passive binding and the mPACA winning guarantees. If the client has no binding state, then note that the authentication oracle **Auth-C** will fail and so will the **rChall** oracle. Otherwise, the reduction can pinpoint the unique token and choose an unused PlA registration session to use its own PlA registration oracle and obtain a registration trace. The first message in this trace is programmed into the output of **rChall**.
- when the adversary calls **rResp**, it is either the case that it is delivering the message to the unique mPACA oracle associated with some prior registration query or not. If not, then the mPACA validate condition will fail and there is nothing to do. Otherwise, the reduction maps the j-th token session in the composed model to the token session in the PlA model that it prehemptively chose to obtain the registration trace. The second message of the passive registration trace is programmed in the output of the oracle.
- when the adversary calls **rCompl**, the reduction simply needs to check if the message received comes from an mPACA token session that is linked with the PlA token session that is the registration partner of the server session that was involved in the initial rChall oracle.

```
rResp $(T, j, j', m_{\mathrm{rcom}}, t_{\mathrm{cl}}, d)$:
 1: if $\tilde{\pi}^{j}_{r,T} \neq \perp$ or $T.\mathrm{gid} = \perp$ then return $\perp$
 2: status $\xleftarrow{\$}$ Validate-T$(T, j', m_{\mathrm{rcom}}, t_{\mathrm{cl}}, d)$
 3: if status $\neq$ accepted then return $\perp$
 4: if Token-Win-SUF-t$(T, j', m_{\mathrm{rcom}}, t_{\mathrm{cl}}, d) = 1$ then
 5:     game ends returning 0
 6: $(m_{\mathrm{rrsp}}, \mathrm{rc}_T, \mathrm{cid}, \mathrm{sid}, \mathrm{agCon}) \xleftarrow{\$}$ rRsp$(\tilde{\pi}^{j}_{r,T}, m_{\mathrm{rcom}})$
 7: $(m_{\mathrm{rrsp}}, t_{\mathrm{tk}}) \xleftarrow{\$}$ Auth-T$(T, j', m_{\mathrm{rrsp}})$
 8: $\mathcal{L}_{\text{pla-paca-T}} \leftarrow \mathcal{L}_{\text{pla-paca-T}} \cup \{(\mathrm{reg}, T, j, j')\}$
 9: return $(m_{\mathrm{rrsp}}, t_{\mathrm{tk}})$
rCompl $(S, i, m_{\mathrm{rcl}}, m_{\mathrm{rrsp}}, \mathrm{gid}, t_{\mathrm{tk}})$:
 1: if $\tilde{\pi}^{i}_{r,S} = \perp$ or $\tilde{\pi}^{i}_{r,S}.\mathrm{st}_{\mathrm{exe}} \neq$ running or $G[\mathrm{gid}] = \perp$ then return $\perp$
 2: Retrieve $(\mathrm{reg}, S, i, C, k)$ from $\mathcal{L}_{\text{pla-paca-S}}$
 3: status $\leftarrow$ Validate-C$(C, k, m_{\mathrm{rrsp}}, t_{\mathrm{tk}})$
 4: if status $\neq$ accepted then return $\perp$
 5: if Client-Win-SUF-t$(C, k, m_{\mathrm{rcom}}, t_{\mathrm{tk}}) = 1$ then
 6:     game ends returning 0
 7: $(d, \mathrm{rc}_S, \mathrm{cid}, \mathrm{sid}, \mathrm{agCon}) \xleftarrow{\$}$ rVrfy$(\tilde{\pi}^{i}_{r,S}, m_{\mathrm{rcl}}, m_{\mathrm{rrsp}}, G[\mathrm{gid}].\mathrm{gpars})$
 8: if $d = 1$ then
 9:     $\mathcal{L}_{\mathrm{reg}} \leftarrow \mathcal{L}_{\mathrm{reg}} \cup \{(S, i, \mathrm{gid}, \mathrm{cid})\}$
10: return $d$

aResp $(T, j, j', m_{\mathrm{acom}}, t_{\mathrm{cl}}, d)$:
 1: if $\tilde{\pi}^{j}_{a,T} \neq \perp$ or $T.\mathrm{gid} = \perp$ then return $\perp$
 2: status $\xleftarrow{\$}$ Validate-T$(T, j', m_{\mathrm{acom}}, t_{\mathrm{cl}}, d)$
 3: if status $\neq$ accepted then return $\perp$
 4: if Token-Win-SUF-t$(T, j', m_{\mathrm{acom}}, t_{\mathrm{cl}}, d) = 1$ then
 5:     game ends returning 0
 6: $(m_{\mathrm{arsp}}, \mathrm{rc}_T, \mathrm{cid}, \mathrm{sid}, \mathrm{agCon}) \xleftarrow{\$}$ aRsp$(\tilde{\pi}^{j}_{a,T}, m_{\mathrm{acom}})$
 7: $(m_{\mathrm{arsp}}, t_{\mathrm{tk}}) \xleftarrow{\$}$ Auth-T$(T, j', m_{\mathrm{arsp}})$
 8: $\mathcal{L}_{\text{pla-paca-T}} \leftarrow \mathcal{L}_{\text{pla-paca-T}} \cup \{(\mathrm{auth}, T, j, j')\}$
 9: return $(m_{\mathrm{arsp}}, t_{\mathrm{tk}})$
aCompl $(S, i, m_{\mathrm{acl}}, m_{\mathrm{arsp}}, t_{\mathrm{tk}})$:
 1: if $\tilde{\pi}^{i}_{a,S} = \perp$ or $\tilde{\pi}^{i}_{a,S}.\mathrm{st}_{\mathrm{exe}} \neq$ running then return $\perp$
 2: Retrieve $(\mathrm{auth}, S, i, C, k)$ from $\mathcal{L}_{\text{pla-paca-S}}$
 3: status $\leftarrow$ Validate-C$(C, k, m_{\mathrm{arsp}}, t_{\mathrm{tk}})$
 4: if status $\neq$ accepted then return $\perp$
 5: if Client-Win-SUF-t$(C, k, m_{\mathrm{acom}}, t_{\mathrm{tk}}) = 1$ then
 6:     game ends returning 0
 7: $(d, \mathrm{rc}_S, \mathrm{cid}, \mathrm{sid}, \mathrm{agCon}) \xleftarrow{\$}$ aVrfy$(\tilde{\pi}^{i}_{a,S}, m_{\mathrm{acl}}, m_{\mathrm{arsp}})$
 8: if $d = 1$ and win-ua $= 0$ then
 9:     win-ua $\leftarrow$ Win-ua$(S, i, \mathrm{cid})$
10: return $d$
```

Figure 22: First hop of the composed model proof. Code in red represents the changes to the code correspoding to the first hop. The changes in Win-ua are redundant, because if the game reaches this function, then it must have passed through mPACA and triggered the same conditions inside Client-Win-SUF-t. We show them only for keeping changes consistent.

The remaining part of the behavior of B_2 is a simple reduction where it uses its own oracles to answer the queries placed by $\mathcal{A}$.

At this point the composed model adversary can only win if it breaks a composed-model specific condition: the server accepts an authentication run, but the bound PlA oracles are not residing in the token that is bound to the correct mPACA client.

This can only happen when at least one of the conditions in lines 27, 28, 37 or 38 from Fig. 23 is true, which requires that the client session bound to the token that completed the registration/authentication run is not the same as the client session linked to the server session that finished that registration/authentication run. Also note, if these winning conditions were activated, it is at a point in the experiment where we can always find the unique relationship between registration and authentication sessions of servers and tokens, via the Partnerships method. We now explain why this cannot occur.

The condition in line 27 models the rogue key attack scenario: the adversary is able to register a key generated by a token PlA session that is not protected by the mPACA session of the client that is connected to the server, and this mPACA session is not compromised. Note that for the server to have accepted the registered key, the client connected to it must have accepted the message containing that key. However, this would have implied an mPACA break which we excluded in hop 1.

```
Win-ua $(S, i, \mathsf{cid})$:
 1: if $\exists(S_1, i_1, ph_1) \neq (S_2, i_2, ph_2)$ s.t. $\tilde{\pi}^{i_1}_{ph_1,S_1}.\mathsf{sid} = \tilde{\pi}^{i_2}_{ph_2,S_2}.\mathsf{sid} \neq \perp$ then game ends
      returning 0
 2: if $\exists(T_1, j_1, ph_1) \neq (T_2, j_2, ph_2)$ s.t. $\tilde{\pi}^{j_1}_{ph_1,T_1}.\mathsf{sid} = \tilde{\pi}^{j_2}_{ph_2,T_2}.\mathsf{sid} \neq \perp$ then game ends
      returning 0
 3: if $\exists(C_1, k_1), (C_2, k_2)$ $s.t.$ $(C_1, j_1) \neq (C_2, j_2)$ and $\pi^{k_1}_{C_1}.\mathsf{st}_\mathsf{exe} = \pi^{k_2}_{C_2}.\mathsf{st}_\mathsf{exe} = \mathsf{bindDone}$
      and $\pi^{k_1}_{C_1}.\mathsf{sid} = \pi^{k_2}_{C_2}.\mathsf{sid}$ then return 1
 4: if $\exists(T_1, j'_1), (T_2, j'_2)$ $s.t.$ $(T_1, j'_1) \neq (T_2, j'_2)$ and $\pi^{j'_1}_{T_1}.\mathsf{st}_\mathsf{exe} = \pi^{j'_2}_{T_2}.\mathsf{st}_\mathsf{exe} = \mathsf{bindDone}$
      and $\pi^{j'_1}_{T_1}.\mathsf{sid} = \pi^{j'_2}_{T_2}.\mathsf{sid}$ then return 1
 5:
 6: if $\exists(T_1, j'_1), (T_2, j'_2)$ $s.t.$ $(T_1, j'_1) \neq (T_2, j'_2)$ and $\pi^{j'_1}_{T_1}.\mathsf{st}_\mathsf{exe} = \pi^{j'_2}_{T_2}.\mathsf{st}_\mathsf{exe} = \mathsf{bindDone}$
      and $\pi^{j'_1}_{T_1}.\mathsf{sid} = \pi^{j'_2}_{T_2}.\mathsf{sid}$ then game ends returning 0
 7:
 8: if $\exists(S_1, i_1) \neq (S_2, i_2)$ s.t. $\tilde{\pi}^{i_1}_{r,S_1}.\mathsf{cid} = \tilde{\pi}^{i_2}_{r,S_2}.\mathsf{cid} \neq \perp$ then game ends returning 0
 9:
10: if $\exists(S', i', ph'), (T', j', ph')$ s.t. $\tilde{\pi}^{i'}_{ph',S'}.\mathsf{sid} = \tilde{\pi}^{j'}_{ph',T'}.\mathsf{sid} \neq \perp$ and $(S', T') \notin \mathcal{L}_\mathsf{corr}$
      and $T'.\mathsf{gid} \notin \mathcal{L}_\mathsf{corrG}$ and $\tilde{\pi}^{i'}_{ph',S'}.\mathsf{agCon} \neq \tilde{\pi}^{j'}_{ph',T'}.\mathsf{agCon}$ then game ends returning 0
11:
12: $(\mathsf{gid}, i', T, j', j) \leftarrow \mathsf{Partnerships}(S, i, \mathsf{cid})$
13: if $\mathsf{gid} = \perp$ or $(\mathsf{gid} \notin \mathcal{L}_\mathsf{corrG}$ and $(T = \perp$ or $T.\mathsf{gid} \neq \mathsf{gid}))$ then game ends returning 0
14: else
15:     if $\mathsf{gid} \notin \mathcal{L}_\mathsf{corrG}$ then
16:         Retrieve $(\mathsf{reg}, T, j', l')$ from $\mathcal{L}_\mathsf{pla\text{-}paca\text{-}T}$
17:         Retrieve $(\mathsf{reg}, S, i', C_S, k_S)$ from $\mathcal{L}_\mathsf{pla\text{-}paca\text{-}S}$
18:         $(C_T, k_T) \leftarrow \mathsf{tokenBindPartner}(T, l')$
19:         if $(C_S, k_S) \neq (C_T, k_T)$ then
20:             $(T_S, m) \leftarrow \mathsf{clientBindPartner}(C_S, k_S)$
21:             if $\pi^{k_S}_{C_S}.\mathsf{compromised} = \mathsf{false}$ and $\pi^{m}_{T_S}.\mathsf{pinCorr} = \mathsf{false}$ then return 1
22:             if $((C_T, k_T) = (\perp, \perp)$ or $\pi^{k_T}_{C_T}.\mathsf{compromised} = \mathsf{false})$ and $\pi^{l'}_{T}.\mathsf{pinCorr} = \mathsf{false}$
                then return 1
23:         if $(S, T) \notin \mathcal{L}_\mathsf{corr}$ and $j = \perp$ then game ends returning 0
24:         else if $(S, T) \notin \mathcal{L}_\mathsf{corr}$ then
25:             Retrieve $(\mathsf{auth}, T, j, l)$ from $\mathcal{L}_\mathsf{pla\text{-}paca\text{-}T}$
26:             Retrieve $(\mathsf{auth}, S, i, C'_S, k'_S)$ from $\mathcal{L}_\mathsf{pla\text{-}paca\text{-}S}$
27:             $(C'_T, k'_T) \leftarrow \mathsf{tokenBindPartner}(T, l)$
28:             if $(C'_S, k'_S) \neq (C'_T, k'_T)$ then
29:                 $(T'_S, m') \leftarrow \mathsf{clientBindPartner}(C_S, k_S)$
30:                 if $\pi^{k'_S}_{C'_S}.\mathsf{compromised} = \mathsf{false}$ and $\pi^{m'}_{T'_S}.\mathsf{pinCorr} = \mathsf{false}$ then return 1
31:                 if $((C'_T, k'_T) = (\perp, \perp)$ or $\pi^{k'_T}_{C'_T}.\mathsf{compromised} = \mathsf{false})$ and $\pi^{l}_{T}.\mathsf{pinCorr} = \mathsf{false}$
                    then return 1
32: return 0
```

Figure 23: Second hop of the composed model proof. Code in red represents the changes to the code correspoding to the second hop.

The condition in line 28 models the scenario where $\mathcal{A}$ is successful in registering a key generated by a token that is out of its control: the adversary, controlling a possibly compromised client that is connected to the server, registers a key generated by a token PlA session in an uncorrupted token mPACA session that has no relation to the client controlled by the adversary. This would imply the adversary was able to break mPACA by breaking into this token and convincing it to answer a response request, which we excluded in hop 1.

The justification for the winning conditions in lines 37 and 38 being unreachable is the same as above, but these lines correspond to different practical attack scenarios that are excluded by our proof. In line 37, the adversary would be hijacking an authentication session established via an uncorrupted client and authenticating using its own token. In line 38, the adversary is able to break into the user's token and impersonate the user in an authenticated session that it controls.

This leaves only one last option for $\mathcal{A}$ to try to win, which is through the conditions in lines 3 and 4. However, these are also never reached, since they mean that $\mathcal{A}$ wins by breaking the mPACA guarantees regarding uniqueness of sid values in client and token sessions, which is not also possible after hop 1.

Therefore, at this point $\mathcal{A}$ can never win the game, and the proof is concluded. □

Theorem 9. *(Attestation* basic*) If there exists an adversary $\mathcal{A}$ against the composed security of an mPACA protocol mPACA and a PlA protocol PlA, then there exist adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ such that:*

$$\mathsf{Adv}^{ua}_{PlA+mPACA}(\mathcal{A}) \leq \mathsf{Adv}^{SUF\text{-}t}_{mPACA}(\mathcal{B}_1) + \mathsf{Adv}^{pla\text{-}auth}_{PlA}(\mathcal{B}_2)\ .$$

The proof of this theorem is similar to the previous one, only that the reduction to PlA security is now simpler given the adaptive power of the adversary.

Note that the results we give in Section 4 for the mPACA security of CTAP 2.1+ and in Appendix G for the security of WebAuthn in various attestation modes imply that these composed security results apply to this improved version of FIDO2. We next discuss how our results capture rogue key attacks.

Rogue key attacks. The rogue key attack against the current instantiation of FIDO2 is possible because the client has no way of verifying the origin of a token response (in CTAP 2.1) *and* because the server may not have any information that uniquely identifies the token from which is expects a response (in WebAuthn). That is indeed the case for the most common attestation modes None, Self and Basic.

To see how our result addresses rogue key attacks above, in *all* attestation modes, consider the scenario where an active attacker is trying to launch a rogue key attack against the composed PlA+mPACA protocol, but it does not have the ability to corrupt the client that the user is relying on. Then, our results above guarantee that the server will only accept a credential generated by the token that is uniquely bound to that client, which in turn is uniquely bound by a TLS connection to the server. As soon as the client is under the adversary's control this guarantee no-longer holds, and rogue key attacks can take place. We note that this is only true for the upgraded version of mPACA that we proposed in this paper. We discuss next what these results mean for the current version of FIDO2.

Composed security for current FIDO2. When considering the current version of FIDO2 we can no longer rely on its PACA component CTAP 2.1 to resolve the problem of rogue key attacks. However, we can consider a weaker security model in which composed security holds: *trust on first use.* In this setting one assumes that the composed protocol adversary is fully passive during registration, as in [6].

However, even in this case, the adversary can try to take advantage of the lack of authentication in messages going from the authenticator back to the client that is bound to the server. The composed model guarantee for current FIDO2 is therefore much weaker than what we have presented above. We describe the implications in detail next.

First of all, one cannot guarantee that the server only accepts an authentication response that comes from the token that is PACA-bound to the client. Indeed, the attacker could potentially convince the client to send back to the server a response that comes from his own maliciously-controlled token, thereby leading the server to log-in the user under a different account. The implications of such an attack could be similar to those of Cross-Site Request Forgery (CSRF) attacks.[12]

Second, even though such an attack is still possible, the role of PACA in composed FIDO2 security is still relevant: it guarantees that, if the user's token is not compromised and only interacts with honest clients, then the attacker cannot break into the token and impersonate the user. Formally this can be captured by taking the approach in [6] to composed model security: one requires that the attacker cannot break PlA security and, furthermore, that tokens bound to honest clients only issue responses to PlA authentication requests if these requests come from their unique PACA partner.

Finally, we remark that we could strengthen the composed model guarantees for current FIDO2 with a different use of credential identifiers. Indeed, assuming *trust on first use*, the server will record a unique credential identifier cid for each credential. Hence, if this is associated with a server-side user identifier, the server could potentially impose a priori a cid when authenticating the user. In this case, the attack we described above will not work, as the attacker's credential will be rejected because it does not match the cid that the server is looking for. We expect that, in this setting, FIDO2 meets the stronger notion of composed security for authentication runs we propose here, but we do not pursue this line of analysis because this does not seem to be the common use case for cids.

J Formal CTAP Privacy Attacks

Figure 24 describes the adversary $\mathcal{A}$ that breaks PACA privacy (defined in Section 5.1) of CTAP 2.1 and CTAP 2.1+ by taking advantage of the reuse of ECDH shares on the authenticator side. Since each ECDH share pk_T is sampled randomly, $\mathcal{A}$ will identify the correct token with probability $1 - 1/q$, where q is the prime order of the underlying ECDH group (from curve P-256).

```
$\mathcal{A}^{\text{priv}}_{\text{CTAP 2.1/CTAP 2.1+}}(\lambda)$:
For some distinct tokens $T_0, T_1$, some clients $C_0, C_1$, and some user $U$
1: Phase 1: $\text{trans}_0 \leftarrow \textbf{Setup}(T_0, 1, C_0, 1, U)$, $\text{trans}_1 \leftarrow \textbf{Setup}(T_1, 1, C_1, 1, U)$
2: Extract token $T$'s DH share $pk_0$ from $\text{trans}_0$ and DH share $pk_1$ from $\text{trans}_1$
3: Phase 2: Outputs $T_0, T_1, C_0, C_1, U, U$
4: Phase 3: $\text{trans} \leftarrow \textbf{Bind}\text{-LEFT}_{T_0,C_0}(2, 2)$
5: Extract token $T$'s DH share $pk$ from trans
6: if $pk = pk_0$ then
7:     return 0
8: else
9:     return 1
```

Figure 24: Privacy adversary for CTAP 2.1 and CTAP 2.1+.

[12]https://owasp.org/www-community/attacks/csrf