# Privacy and Security of FIDO2 Revisited

Manuel Barbosa[1,4], Alexandra Boldyreva[2], Shan Chen[3], Kaishuo Cheng[2], and Luís Esquível[1]

[1] Universidade do Porto (FCUP) & INESC TEC, Portugal
mbb@fc.up.pt, luis.esquivel.costa@gmail.com
[2] Georgia Institute of Technology, USA
{sasha, kcheng89}@gatech.edu
[3] Southern University of Science and Technology, China[*]
dragoncs16@gmail.com
[4] Max Planck Institute for Security and Privacy, Germany

**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

---

[*] Affiliated with both the Research Institute of Trustworthy Autonomous Systems and the Department of Computer Science and Engineering

# 1  Introduction

The fast-growing adoption of FIDO2 [13], 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[5], a community of stakeholders that manages the specifications of the protocol and promotes its adoption.

The basic flows of FIDO2 are shown in Fig. 1, inspired by the corresponding figure in [2]. At its core, FIDO2 is composed of two sub-protocols: WebAuthn (W3C's Web Authentication) [23,24] and CTAP2 (Client to Authenticator Protocol version 2.x) [12,1].

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.
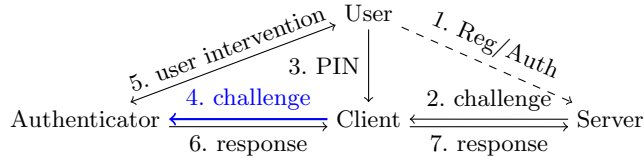
---

[5] `https://fidoalliance.org`

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

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 [13]. The privacy properties of FIDO2 have been formally analyzed in [15,16,6] 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 [15] and, more recently, on the overall FIDO2 as composed by WebAuthn and CTAP [2,5]. 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.* [16] 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.* [6] 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[6]. It was acknowledged in [16] 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

---

[6] Kepkowski *et al.* [18] 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.

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 [3] 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 [8] 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 [2] 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 [5] considered only attackers that are passive during registration, which is not realistic as the aforementioned attacks demonstrate. The work [6] considered only the WebAuthn protocol, which means that the security of client-token communications is not considered. Finally, the full version of [2] 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 [16,6], 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

4

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 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 [5] in the form of an attack that breaks the protocol in the security model for authentication adopted by [5]. 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 [2,5] 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.[7] We give a detailed proof of FIDO2 authentication security in this model that addresses the shortcomings we

---

[7] 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

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 [3,8]. 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.

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 $\epsilon$. 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 $\bot$ to denote any uninitialized variable, as well as the return value from an algorithm that executes incorrectly.

**Security Games.** We follow closely [5] 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

---

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.

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. [2] 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. [5], 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. [5] focused on settings where no long-term attestation keys are used and hence assumed passive attackers in registration flows. Later, Bindel et al. [6] continued the work of [5], focusing only on WebAuthn, but considered various attestation modes and extended the analysis to privacy guarantees. More detailed background is summarized next.

**WebAuthn and PlA.** We depict the WebAuthn protocol (with attestation mode Basic) in Figure 2 by following the descriptions in [5,6]. As in [6], our WebAuthn description captures the cryptographic core of both the current stable version WebAuthn 2 [23] and a working draft for WebAuthn 3 [24], so we henceforth simply refer to this protocol as WebAuthn[8].

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

---

[8] WebAuthn 3 [24] 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 [6]) is not changed.
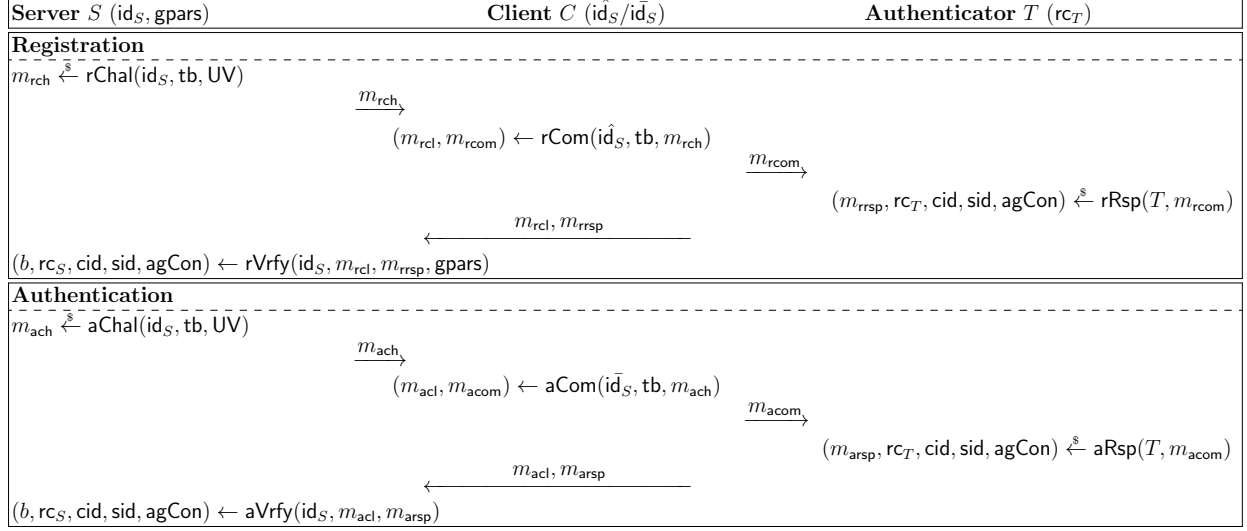
| Server $S$ ($\mathsf{id}_S$, gpars) | Client $C$ ($\hat{\mathsf{id}}_S/\bar{\mathsf{id}}_S$) | Authenticator $T$ ($\mathsf{rc}_T$) |
|---|---|---|
| **Registration** | | |
| $m_\mathsf{rch} \xleftarrow{\$} \mathsf{rChal}(\mathsf{id}_S, \mathsf{tb}, \mathsf{UV})$ | | |
| $\xrightarrow{\quad m_\mathsf{rch} \quad}$ | | |
| | $(m_\mathsf{rcl}, m_\mathsf{rcom}) \leftarrow \mathsf{rCom}(\hat{\mathsf{id}}_S, \mathsf{tb}, m_\mathsf{rch})$ | |
| | $\xrightarrow{\quad m_\mathsf{rcom} \quad}$ | |
| | | $(m_\mathsf{rrsp}, \mathsf{rc}_T, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \xleftarrow{\$} \mathsf{rRsp}(T, m_\mathsf{rcom})$ |
| | $\xleftarrow{\quad m_\mathsf{rcl}, m_\mathsf{rrsp} \quad}$ | |
| $(b, \mathsf{rc}_S, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \leftarrow \mathsf{rVrfy}(\mathsf{id}_S, m_\mathsf{rcl}, m_\mathsf{rrsp}, \mathsf{gpars})$ | | |
| **Authentication** | | |
| $m_\mathsf{ach} \xleftarrow{\$} \mathsf{aChal}(\mathsf{id}_S, \mathsf{tb}, \mathsf{UV})$ | | |
| $\xrightarrow{\quad m_\mathsf{ach} \quad}$ | | |
| | $(m_\mathsf{acl}, m_\mathsf{acom}) \leftarrow \mathsf{aCom}(\bar{\mathsf{id}}_S, \mathsf{tb}, m_\mathsf{ach})$ | |
| | $\xrightarrow{\quad m_\mathsf{acom} \quad}$ | |
| | | $(m_\mathsf{arsp}, \mathsf{rc}_T, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \xleftarrow{\$} \mathsf{aRsp}(T, m_\mathsf{acom})$ |
| | $\xleftarrow{\quad m_\mathsf{acl}, m_\mathsf{arsp} \quad}$ | |
| $(b, \mathsf{rc}_S, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \leftarrow \mathsf{aVrfy}(\mathsf{id}_S, m_\mathsf{acl}, m_\mathsf{arsp})$ | | |

**Fig. 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 (gpars, rc) $\xleftarrow{\$}$ 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 $\mathsf{rc}_T$ is initialized as rc, which contains the attestation private key ak; the server inputs its identity $\mathsf{id}_S$ and attestation info gpars. For attestation modes None and Self, both $\mathsf{rc}_T$ and gpars are empty.

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 [2,5,6] 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 [6] 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 [2], which is more restrictive and therefore makes the model stronger. The details are in Appendix G, where we discuss the model differences to [6] and explain why the proofs in [6] 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 [12]. 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.

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[9], 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 [2] 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 [5] refines the one in [2] 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

---

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

| Client $C$ ($\mathsf{pin}_U$) | | Authenticator $T$ |
|---|---|---|
| **Reboot** | | |
| | | $\mathsf{authPowerUp\text{-}T}(\mathsf{st}_T)$ |
| **Setup** | | |
| | $\xleftarrow{\quad\mathsf{info}\quad}$ | $\mathsf{info} \leftarrow \mathsf{getInfo\text{-}T}(\pi_T^i)$ |
| $\mathsf{puvProtocol} \xleftarrow{\$} \mathsf{obtainSharedSecret\text{-}C\text{-}start}(\pi_C^j, \mathsf{info})$ | | |
| | $\xrightarrow{\mathsf{puvProtocol}}$ | $pk_T \xleftarrow{\$} \mathsf{obtainSharedSecret\text{-}T}(\pi_T^i, \mathsf{puvProtocol})$ |
| $c \xleftarrow{\$} \mathsf{obtainSharedSecret\text{-}C\text{-}end}(\pi_C^j, pk_T)$ | $\xleftarrow{\quad pk_T \quad}$ | |
| $(c_p, t_p) \xleftarrow{\$} \mathsf{setPIN\text{-}C}(\pi_C^j, \mathsf{pin}_U)$ | $\xrightarrow{c, c_p, t_p}$ | $\mathsf{status} \leftarrow \mathsf{setPIN\text{-}T}(\pi_T^i, \mathsf{puvProtocol}, c, c_p, t_p)$ |
| **Bind** | | |
| **Bind-C** | | **Bind-T** |
| if $\pi_C^j.\mathsf{st}_{\mathsf{exe}} = \bot$: | $\xleftarrow{\quad\mathsf{info}\quad}$ | $\mathsf{info} \leftarrow \mathsf{getInfo\text{-}T}(\pi_T^i)$ |
| $\quad \mathsf{puvProtocol} \xleftarrow{\$} \mathsf{obtainSharedSecret\text{-}C\text{-}start}(\pi_C^j, \mathsf{info})$ | | |
| | $\xrightarrow{\mathsf{puvProtocol}}$ | if $\pi_T^i.\mathsf{st}_{\mathsf{exe}} = \bot$: |
| | $\xleftarrow{\quad pk_T \quad}$ | $\quad pk_T \leftarrow \mathsf{obtainSharedSecret\text{-}T}(\pi_T^i, \mathsf{puvProtocol})$ |
| if $\pi_C^j.\mathsf{st}_{\mathsf{exe}} = \mathsf{waiting}$ : | | |
| $\quad c \xleftarrow{\$} \mathsf{obtainSharedSecret\text{-}C\text{-}end}(\pi_C^j, pk_T)$ | | |
| $\quad c_{ph} \xleftarrow{\$} \mathsf{obtainPinUvAuthToken\text{-}C\text{-}start}(\pi_C^j, \mathsf{pin}_U)$ | | |
| | $\xrightarrow{c, c_{ph}}$ | if $\pi_T^i.\mathsf{st}_{\mathsf{exe}} = \mathsf{waiting}$ : |
| | | $\quad c_{pt} \xleftarrow{\$} \mathsf{obtainPinUvAuthToken\text{-}T}(\pi_T^i, \mathsf{puvProtocol}, c, c_{ph})$ |
| if $\pi_C^j.\mathsf{st}_{\mathsf{exe}} = \mathsf{bindStart}$ : | $\xleftarrow{\quad c_{pt} \quad}$ | |
| $\quad \mathsf{obtainPinUvAuthToken\text{-}C\text{-}end}(\pi_C^j, c_{pt})$ | | |
| **Auth-C** | | **Validate-T** |
| if $\pi_C^j.\mathsf{st}_{\mathsf{exe}} = \mathsf{bindDone}$ : | $\xrightarrow{\bar{M}, t}$ | if $\pi_T^i.\mathsf{st}_{\mathsf{exe}} = \mathsf{bindDone}$ : |
| $\quad (M, t) \xleftarrow{\$} \mathsf{auth\text{-}C}(\pi_C^j, M)$ | | $\quad \mathsf{status} \leftarrow \mathsf{validate\text{-}T}(\pi_T^i, M, t, d)$ |
| <span style="color:red">**Validate-C**</span> | | <span style="color:red">**Auth-T**</span> |
| <span style="color:red">if $\pi_C^j.\mathsf{st}_{\mathsf{exe}} = \mathsf{bindDone}$ :</span> | $\xleftarrow{\bar{R}, t}$ | <span style="color:red">if $\pi_T^i.\mathsf{st}_{\mathsf{exe}} = \mathsf{bindDone}$ :</span> |
| <span style="color:red">$\quad \mathsf{status} \leftarrow \mathsf{validate\text{-}C}(\pi_C^j, R, t)$</span> | | <span style="color:red">$\quad (R, t) \xleftarrow{\$} \mathsf{auth\text{-}T}(\pi_T^i, R)$</span> |

**Fig. 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 $\mathsf{obtainSharedSecret\text{-}T}$ execution). Complete descriptions of the algorithms are shown in Appendix B.

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 [5]. 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 [2,5] 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 [2,5] 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 [5] 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 [2] 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 [5], which itself follows [2]. 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. 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 [2] 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 $\pi_T^i$ and $\pi_C^j$ 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 $\mathsf{st}_T$, which contains the associated user identifier $\mathsf{st}_T.\mathsf{user}$, some information about the $\mathsf{pin}$, 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. $\pi_T^i$ and $\pi_C^j$ have a binding state $\mathsf{bs}$, session identifier $\mathsf{sid}$ and execution state $\mathsf{st}_{\mathsf{exe}} \in \{\bot, \mathsf{waiting}, \mathsf{bindStart}, \mathsf{bindDone}\}$. Here $\bot$ indicates that the session (oracle) is not yet initialized, in which case we simply write $\pi_T^i = \bot$ or $\pi_C^j = \bot$. 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 [5] 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 [2], which differs from the model in [5] 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 $\pi_T^i$.pinCorr variable determining if the pin associated with token $T$ has been corrupted, and each client session has a $\pi_C^j$.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 [5]. 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 $\pi_C^j$ and $\pi_T^i$ 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 $\pi_C^j$ and $\pi_T^i$, 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 $\pi_C^j$.bs and permanently marks this session as compromised. Reboot on a token session $\pi_T^i$ calls Reboot subprotocol on $T$ and marks all of $T$'s sessions as invalid. Auth-C authorizes a message $M$ on a client session $\pi_C^j$ 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 $\pi_T^i$ and outputs a boolean response. Finally, Send-Bind-T allows $\mathcal{A}$ to send a message $m$ to a token session $\pi_T^i$ to initiate, continue or complete Bind; in the latter case, it also invalidates all of $T$'s previous sessions, including $\pi_T^i$ if the query caused $T$ to reboot.

**Session partnership.** We say sessions $\pi_C^j$ and $\pi_T^i$ 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 $\mathsf{Expt}_{\mathsf{PACA}}^{\mathsf{SUF\text{-}t}}$ wins if it gets a message-tag pair $(M, t)$ accepted by a token session $\pi_T^i$ 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* $\pi_T^i$ 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 Fig. 4, which is checked whenever the adversary queries Validate-T.

13

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

$\mathsf{Expt}_{\mathsf{PACA}}^{\mathsf{SUF\text{-}t}}$ $(\mathcal{A})$:
1: $\mathcal{L}_{\mathsf{authC}} \leftarrow \emptyset$
2: win-SUF-t $\leftarrow 0$
3: $() \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(1^\lambda)$
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}.\mathsf{st}_{\mathsf{exe}} = \pi_{C_2}^{j_2}.\mathsf{st}_{\mathsf{exe}} = $ bindDone
   **and** $\pi_{C_1}^{j_1}.\mathsf{sid} = \pi_{C_2}^{j_2}.\mathsf{sid}$ **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}.\mathsf{st}_{\mathsf{exe}} = \pi_{T_2}^{i_2}.\mathsf{st}_{\mathsf{exe}} = $ bindDone
   **and** $\pi_{T_1}^{i_1}.\mathsf{sid} = \pi_{T_2}^{i_2}.\mathsf{sid}$ **then return** 1
6:
7: $(C, j) \leftarrow$ tokenBindPartner$(T, i)$
8: **if** $(C, j, M, t) \notin \mathcal{L}_{\mathsf{authC}}$ **then**
9: │ **if** $(C, j) = (\bot, \bot)$ **or** $\pi_C^j.\mathsf{compromised} = $ false **then**
10: │ │ **if** $\pi_T^i.\mathsf{pinCorr} = $ false **then**
11: │ │ └ **return** 1
12: **return** 0

---

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

$\mathsf{Expt}_{\mathsf{mPACA}}^{\mathsf{SUF\text{-}t}}$ $(\mathcal{A})$:
1: $\mathcal{L}_{\mathsf{authC}}, \mathcal{L}_{\mathsf{authT}} \leftarrow \emptyset$
2: win-SUF-t $\leftarrow 0$
3: $() \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(1^\lambda)$
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}.\mathsf{st}_{\mathsf{exe}} = \pi_{C_2}^{j_2}.\mathsf{st}_{\mathsf{exe}} = $ bindDone
   **and** $\pi_{C_1}^{j_1}.\mathsf{sid} = \pi_{C_2}^{j_2}.\mathsf{sid}$ **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}.\mathsf{st}_{\mathsf{exe}} = \pi_{T_2}^{i_2}.\mathsf{st}_{\mathsf{exe}} = $ bindDone
   **and** $\pi_{T_1}^{i_1}.\mathsf{sid} = \pi_{T_2}^{i_2}.\mathsf{sid}$ **then return** 1
4:
5: $(T, i) \leftarrow$ clientBindPartner$(C, j)$
6: **if** $(T, i, M, t) \notin \mathcal{L}_{\mathsf{authT}}$ **then**
7: │ **if** $\pi_C^j.\mathsf{compromised} = $ false **then**
8: │ │ **if** $\pi_T^i.\mathsf{pinCorr} = $ false **then**
9: │ │ └ **return** 1
10: **return** 0

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

NewT $(T, \text{initialData})$:
1: **if** $\text{st}_T \neq \perp$ **then return** $\perp$
2: $\text{st}_T.\text{initialData} \leftarrow \text{initialData}$
3: $\text{Reboot}(\text{st}_T)$
4: **return**

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

Reboot $(T)$:
1: **if** $\text{st}_T = \perp$ **then return** $\perp$
2: **for all** $i$ $s.t.$ $\pi_T^i \neq \perp$ **do**
3: $\quad \lfloor \ \pi_T^i.\text{isValid} \leftarrow \text{false}$
4: $\widetilde{\text{Reboot}}(\text{st}_T)$
5: **return**

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

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

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

NewU $(U)$:
1: **if** $\mathcal{L}_{\text{valid}}[U] = \perp$ **then**
2: $\quad \mid \ \text{pin} \overset{\$}{\leftarrow} \mathcal{P}$
3: $\quad \mid \ \mathcal{L}_{\text{valid}}[U] \leftarrow \text{pin}$
4: $\quad \lfloor \ \mathcal{L}_{\text{corrupt}}[U] \leftarrow \text{false}$
5: **return**

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

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

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

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

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

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

**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 $\mathsf{Expt}_{\mathsf{mPACA}}^{\mathsf{SUF\text{-}t}}$ (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.

**Advantage measures.** For protocol $\mathsf{prot} \in \{\mathsf{PACA}, \mathsf{mPACA}\}$, the authentication advantage of $\mathcal{A}$ is defined as:
$$\mathsf{Adv}_{\mathsf{prot}}^{\mathsf{SUF\text{-}t}}(\mathcal{A}) = \Pr[\mathsf{Expt}_{\mathsf{prot}}^{\mathsf{SUF\text{-}t}}(\mathcal{A}) = 1] \ .$$

**Differences to [5].** Our security model modifies the Execute and Send-Bind-T oracles from [5] back to the modeling approach of [2], in order to exclude active attacks on clients. This means that Execute captures a full Bind execution between two sessions $\pi_C^j$ and $\pi_T^i$, whereas Send-Bind-T allows the adversary to arbitrarily attempt to bind with a token session $\pi_T^i$. We remove the Send-Bind-C oracle defined in [5], 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 [5], 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 [5] 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 [2], 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 [5, 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 [7]. 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 [2]. 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.[10] Nevertheless we note that, as noted in [8], 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 [2] 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 Fig. 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 [19] 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) [17], the underlying symmetric encryption scheme SKE denotes AES-256 in CBC mode with random IV [9], and the message authentication code MAC denotes HMAC-SHA-256 with 256-bit keys [4]. A query to Send-Bind-T on a session $\pi_T^i$ is considered to be *active*, if $\mathcal{A}$ delivers a DH share $c$ and ciphertext $c_{ph}$ to $\pi_T^i$ 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:*

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

---

[10] 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.

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 Appendix L.

## 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 [6] 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 [5], 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 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 [5] 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 [3] 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

| Server/Client $(vk, pt_c)$ | Attacker $(pt_{at})$ | Tokens |
|---|---|---|

$ch \leftarrow \mathsf{rChal}$

User Token (ak)

$t_c \leftarrow \mathsf{auth\text{-}C}(pt_c, ch)$ $\quad \xrightarrow{\;ch\;}_{t_c} \quad \xrightarrow{\;ch\;}_{t_c} \quad \xrightarrow{\;ch\;}_{t_c}$ *token checks user presence*

$\xleftarrow{(pk, \sigma_c)} \quad \sigma_c \leftarrow \mathsf{Sign}(ak, (pk, ch))$

Attacker Token (ak)

$t_{at} \leftarrow \mathsf{auth\text{-}C}(pt_{at}, ch) \quad \xrightarrow{\;ch\;}_{t_{at}}$ *token checks user presence*

$1 \leftarrow \mathsf{Ver}(vk, (pk^\star, ch), \sigma_{at}) \xleftarrow{(pk^\star, \sigma_{at})} \xleftarrow{(pk^\star, \sigma_{at})} \xleftarrow{(pk^\star, \sigma_{at})} \quad \sigma_{at} \leftarrow \mathsf{Sign}(ak, (pk^\star, ch))$
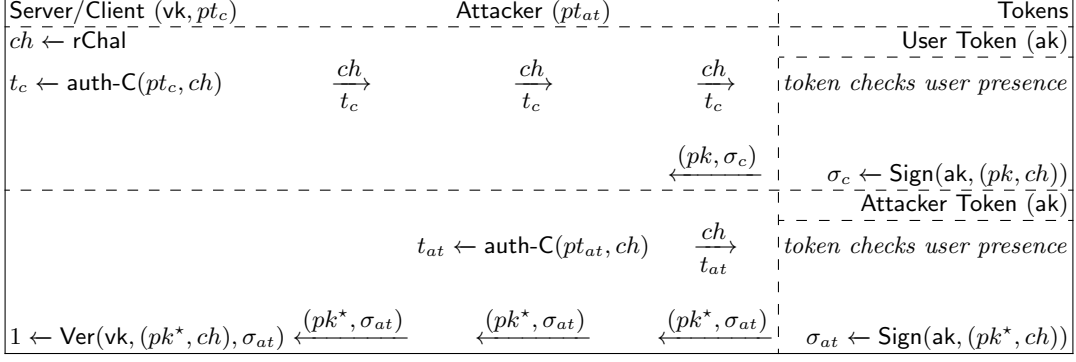
**Fig. 6.** The rogue key attack (against a registration run) in Basic attestation. Right arrows represent commands for requesting a credential and left arrows are the respective responses. Procedures are simplified for better presentation.

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 [3,8], 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 [5] 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 [3] 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 [2], 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.

19

**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 $q_{Send}^{act}$ 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:*

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

We provide the proof sketch in Appendix D.2 and the full code-based proof in Appendix L.

**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 [5] 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 [16,6] 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 $(T_1, C_1)$.

To avoid trivial wins, we follow the PlA privacy models [16,6] 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

$\mathsf{Expt}_{\mathsf{PACA/mPACA}}^{\mathsf{(w)priv}}(\mathcal{A})$:

1: $\mathcal{L}_{\mathsf{corr}} \leftarrow \varnothing, \mathcal{L}_{\mathsf{ch}}^{\mathsf{bd}} \leftarrow \varnothing, \mathcal{L}_{\mathsf{lr}}^{\mathsf{bd}} \leftarrow \varnothing, \mathcal{L}_{\mathsf{ch}}^{\mathsf{op}} \leftarrow \varnothing, \mathcal{L}_{\mathsf{lr}}^{\mathsf{op}} \leftarrow \varnothing, \mathsf{pb} \leftarrow 1,$ $\mathsf{freshDH} \leftarrow 1$
2: $\mathsf{st}_1 \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(1^\lambda) \;//\; \text{Phase 1}$
3: $T_0, T_1, C_0, C_1, U_0, U_1, \mathsf{st}_2 \leftarrow \mathcal{A}(1^\lambda, \mathsf{st}_1) \;//\; \text{Phase 2}$
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: $b' \leftarrow \mathcal{A}^{\mathcal{O}', \mathsf{LEFT}, \mathsf{RIGHT}}(1^\lambda, \mathsf{st}_2) \;//\; \text{Phase 3}$
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 $(T_{\mathsf{ch}}, i) \in \mathcal{L}_{\mathsf{ch}}^{\mathsf{bd}}, (T_{\mathsf{lr}}, j) \in \mathcal{L}_{\mathsf{lr}}^{\mathsf{bd}}$ **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:          $\mathsf{freshDH} \leftarrow 0$
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: Check $\leftarrow T_0.\mathsf{pinCorr} = \mathsf{false}$ **and** $T_1.\mathsf{pinCorr} = \mathsf{false}$ **and** $\mathcal{L}_{\mathsf{lr}}^{\mathsf{bd}} \cap \mathcal{L}_{\mathsf{corr}} = \varnothing$
8: $\mathsf{S} \leftarrow (\mathcal{L}_{\mathsf{ch}}^{\mathsf{bd}} \cap \mathcal{L}_{\mathsf{lr}}^{\mathsf{op}}) \cup (\mathcal{L}_{\mathsf{ch}}^{\mathsf{op}} \cap \mathcal{L}_{\mathsf{lr}}^{\mathsf{bd}}) \cup (\mathcal{L}_{\mathsf{ch}}^{\mathsf{bd}} \cap \mathcal{L}_{\mathsf{lr}}^{\mathsf{bd}})$
9: **if** $\mathsf{S} = \varnothing$ **and** Check **and** $\mathsf{pb} = 1$ **and** $\mathsf{freshDH} = 1$ **then return** $1$
10: **else return** $0$

$\mathsf{InitRL}(T_0, T_1, C_0, C_1, U_0, U_1)$:

1: $b \xleftarrow{\$} \{0,1\}$
2: Initialize LEFT and RIGHT oracles
3: **return** $b$

$\mathbf{Setup\text{-}LEFT}_{T_b, C_b, U_b}(i, j)$:
1: **return** $\mathsf{Setup}(T_b, i, C_b, j, U_b)$
$\mathbf{Setup\text{-}RIGHT}_{T_{1-b}, C_{1-b}, U_{1-b}}(i, j)$:
1: **return** $\mathsf{Setup}(T_{1-b}, i, C_{1-b}, j, U_{1-b})$
$\mathbf{Bind\text{-}LEFT}_{T_b, C_b}(i, j)$:
1: $\mathcal{L}_{\mathsf{lr}}^{\mathsf{bd}} \xleftarrow{\cup} \{(T_0, i), (C_0, j)\}, \mathcal{L}_{\mathsf{lr}}^{\mathsf{bd}} \xleftarrow{\cup} \{(T_1, i), (C_1, j)\}$
2: **return** $\mathsf{Execute}(T_b, i, C_b, j)$
$\mathbf{Bind\text{-}RIGHT}_{T_{1-b}, C_{1-b}}(i, j)$:
1: $\mathcal{L}_{\mathsf{lr}}^{\mathsf{bd}} \xleftarrow{\cup} \{(T_0, i), (C_0, j)\}, \mathcal{L}_{\mathsf{lr}}^{\mathsf{bd}} \xleftarrow{\cup} \{(T_1, i), (C_1, j)\}$
2: **return** $\mathsf{Execute}(T_{1-b}, i, C_{1-b}, j)$

$\mathbf{Send\text{-}LEFT}_{T_b}(i, M)$
1: $\mathcal{L}_{\mathsf{lr}}^{\mathsf{bd}} \xleftarrow{\cup} \{(T_0, i)\}, \mathcal{L}_{\mathsf{lr}}^{\mathsf{bd}} \xleftarrow{\cup} \{(T_1, i)\}$
2: **return** $\mathsf{Send\text{-}Bind\text{-}T}(T_b, i, M)$
$\mathbf{Send\text{-}RIGHT}_{T_{1-b}}(i, M)$
1: $\mathcal{L}_{\mathsf{lr}}^{\mathsf{bd}} \xleftarrow{\cup} \{(T_0, i)\}, \mathcal{L}_{\mathsf{lr}}^{\mathsf{bd}} \xleftarrow{\cup} \{(T_1, i)\}$
2: **return** $\mathsf{Send\text{-}Bind\text{-}T}(T_{1-b}, i, M)$
$\mathbf{Auth\text{-}C\text{-}LEFT}_{C_b}(j, M)$:
1: $\mathcal{L}_{\mathsf{lr}}^{\mathsf{op}} \xleftarrow{\cup} \{(C_0, j)\}, \mathcal{L}_{\mathsf{lr}}^{\mathsf{op}} \xleftarrow{\cup} \{(C_1, j)\}$
2: $(M, t) \leftarrow \mathbf{Auth\text{-}C}(C_b, j, M)$
3: **return** $(t, \pi_{C_b}^j.\mathsf{bs})$
$\mathbf{Auth\text{-}C\text{-}RIGHT}_{C_{1-b}}(j, M)$:
1: $\mathcal{L}_{\mathsf{lr}}^{\mathsf{op}} \xleftarrow{\cup} \{(C_0, j)\}, \mathcal{L}_{\mathsf{lr}}^{\mathsf{op}} \xleftarrow{\cup} \{(C_1, j)\}$
2: $(M, t) \leftarrow \mathbf{Auth\text{-}C}(C_{1-b}, j, M)$
3: **return** $(t, \pi_{C_{1-b}}^j.\mathsf{bs})$
$\mathbf{Validate\text{-}T\text{-}LEFT}_{T_b}(i, M, t, d)$:
1: $\mathcal{L}_{\mathsf{lr}}^{\mathsf{op}} \xleftarrow{\cup} \{(T_0, i)\}, \mathcal{L}_{\mathsf{lr}}^{\mathsf{op}} \xleftarrow{\cup} \{(T_1, i)\}$
2: $d \leftarrow \mathbf{Validate\text{-}T}(T_b, i, M, t, d).$
3: **return** $(d, \pi_{T_b}^i.\mathsf{bs})$
$\mathbf{Validate\text{-}T\text{-}RIGHT}_{T_{1-b}}(i, M, t, d)$:
1: $\mathcal{L}_{\mathsf{lr}}^{\mathsf{op}} \xleftarrow{\cup} \{(T_0, i)\}, \mathcal{L}_{\mathsf{lr}}^{\mathsf{op}} \xleftarrow{\cup} \{(T_1, i)\}$
2: $d \leftarrow \mathbf{Validate\text{-}T}(T_{1-b}, i, M, t, d).$
3: **return** $(d, \pi_{T_{1-b}}^i.\mathsf{bs})$
$\mathbf{Auth\text{-}T\text{-}LEFT}_{T_b}(i, M)$:
1: $\mathcal{L}_{\mathsf{lr}}^{\mathsf{op}} \xleftarrow{\cup} \{(T_0, i)\}, \mathcal{L}_{\mathsf{lr}}^{\mathsf{op}} \xleftarrow{\cup} \{(T_1, i)\}$
2: $(M, t) \leftarrow \mathbf{Auth\text{-}T}(T_b, i, M)$
3: **return** $(t, \pi_{T_b}^i.\mathsf{bs})$
$\mathbf{Auth\text{-}T\text{-}RIGHT}_{T_{1-b}}(i, M)$:
1: $\mathcal{L}_{\mathsf{lr}}^{\mathsf{op}} \xleftarrow{\cup} \{(T_0, i)\}, \mathcal{L}_{\mathsf{lr}}^{\mathsf{op}} \xleftarrow{\cup} \{(T_1, i)\}$
2: $(M, t) \leftarrow \mathbf{Auth\text{-}T}(T_{1-b}, i, M)$
3: **return** $(t, \pi_{T_{1-b}}^i.\mathsf{bs})$
$\mathbf{Validate\text{-}C\text{-}LEFT}_{C_b}(j, M, t, d)$:
1: $\mathcal{L}_{\mathsf{lr}}^{\mathsf{op}} \xleftarrow{\cup} \{(C_0, j)\}, \mathcal{L}_{\mathsf{lr}}^{\mathsf{op}} \xleftarrow{\cup} \{(C_1, j)\}$
2: $d \leftarrow \mathbf{Validate\text{-}C}(C_b, j, M, t, d).$
3: **return** $(d, \pi_{C_b}^j.\mathsf{bs})$
$\mathbf{Validate\text{-}C\text{-}RIGHT}_{C_{1-b}}(j, M, t, d)$:
1: $\mathcal{L}_{\mathsf{lr}}^{\mathsf{op}} \xleftarrow{\cup} \{(C_0, j)\}, \mathcal{L}_{\mathsf{lr}}^{\mathsf{op}} \xleftarrow{\cup} \{(C_1, j)\}$
2: $d \leftarrow \mathbf{Validate\text{-}C}(C_{1-b}, j, M, t, d)$
3: **return** $(d, \pi_{C_{1-b}}^j.\mathsf{bs})$

**Fig. 7.** (m)PACA privacy security experiments. $\mathsf{Expt}_{\mathsf{PACA}}^{\mathsf{priv}}$ for PACA privacy (without red or blue parts), $\mathsf{Expt}_{\mathsf{PACA}}^{\mathsf{wpriv}}$ for PACA weak privacy (without red parts), $\mathsf{Expt}_{\mathsf{mPACA}}^{\mathsf{priv}}$ for mPACA privacy (without blue parts), $\mathsf{Expt}_{\mathsf{mPACA}}^{\mathsf{wpriv}}$ 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 Fig. 5).

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 $\pi_T^i$ and $\pi_C^j$, 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 $\pi^i_{T_0}$ and $\pi^j_{C_0}$ 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 $(\mathcal{L}^{bd}_{ch} \cap \mathcal{L}^{op}_{lr}) \cup (\mathcal{L}^{op}_{ch} \cap \mathcal{L}^{bd}_{lr})$ 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 $\mathcal{L}^{bd}_{ch} \cap \mathcal{L}^{bd}_{lr}$.

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 $\mathsf{Public}(T)$ defined in Section 3. To avoid trivial wins, we require $\mathsf{Public}(T_0) = \mathsf{Public}(T_1)$ 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 $\mathsf{Expt}^{priv}_{mPACA}$ in Figure 7.

**Advantage measures.** For protocol $\mathsf{prot} \in \{\mathsf{PACA}, \mathsf{mPACA}\}$, the privacy advantage of $\mathcal{A}$ is defined as:
$$\mathsf{Adv}^{priv}_{prot}(\mathcal{A}) = |2\Pr[\mathsf{Expt}^{priv}_{prot}(\mathcal{A}) = 1] - 1| .$$

## 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:

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

where $\mathsf{prot} \in \{\mathsf{CTAP\ 2.1}, \mathsf{CTAP\ 2.1+}\}$, $q$ 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 [3]) 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 $\mathsf{prot} \in \{\mathsf{PACA}, \mathsf{mPACA}\}$, 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+, $\mathsf{Public}(T)$ 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 $\mathscr{D}$ with minimum entropy $h_{\mathscr{D}}$, there exist efficient adversaries $\mathcal{B}_1$, $\mathcal{B}_4$, $\mathcal{B}_5$, $\mathcal{B}_6$ such that:*

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

*where $\mathsf{prot} \in \{\mathsf{CTAP\ 2.1}, \mathsf{CTAP\ 2.1+}\}$.*

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.[11] 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 $q_{Send}^{act}$ active queries to Send-Bind-T, for some PIN-sampling distribution $\mathscr{D}$ with minimum entropy $h_{\mathscr{D}}$, there exist adversaries $\mathcal{B}_1$, $\mathcal{B}_4$, $\mathcal{B}_5$, $\mathcal{B}_6$ such that:*

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

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 [6], discuss how we slightly strengthen and generalize it, and define PlA privacy advantage $\mathsf{Adv}_{PlA}^{priv}$. Theorem 7 in Appendix G.3 states the WebAuthn privacy result, for which the proof from [6] still applies. It shows that for any $\mathcal{A}$, $\mathsf{Adv}_{WebAuthn}^{priv}(\mathcal{A}) = 0$, 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 $\mathsf{Expt}_{PlA+PACA}^{com\text{-}priv}$, $\mathsf{Expt}_{PlA+mPACA}^{com\text{-}priv}$ are shown in Fig. 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 Fig. 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

---

[11] Note that CTAP 2.1++ still achieves mPACA authentication security. The only change to the proof given in Appendix D.2 is in bound $|\mathsf{Pr}_2 - \mathsf{Pr}_3|$, 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.

**Fig. 8.** Composed privacy experiments $\mathsf{Expt}^{\mathsf{com\text{-}priv}}_{\mathsf{PIA+mPACA}}$ (with red code) and $\mathsf{Expt}^{\mathsf{com\text{-}priv}}_{\mathsf{PIA+PACA}}$ (without red code). They each also has a "weak" privacy version, $\mathsf{Expt}^{\mathsf{com\text{-}wpriv}}_{\mathsf{PIA+mPACA}}$ and $\mathsf{Expt}^{\mathsf{com\text{-}wpriv}}_{\mathsf{PIA+PACA}}$, which capture (m)PACA weak privacy. $\mathcal{O}$ contains all (m)PACA privacy oracles (Fig. 7) except those Auth and Validate oracles and all PlA oracles (Fig. 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.

composed check is the conjunction of Check-priv-PIA (defined in Appendix G.3) and Check-priv-PACA (defined in Fig. 7).

Fig. 8 also shows experiments $\mathsf{Expt}^{\mathsf{com\text{-}wpriv}}_{\mathsf{PIA+PACA}}$, $\mathsf{Expt}^{\mathsf{com\text{-}wpriv}}_{\mathsf{PIA+mPACA}}$ 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 $\mathsf{notion} \in \{\mathsf{com\text{-}priv}, \mathsf{com\text{-}wpriv}\}$ and $\mathsf{prot} \in \{\mathsf{PIA + PACA}, \mathsf{PIA + mPACA}\}$ 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:*

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

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:*

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

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 ([10,11]) 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 [21] and Mozilla [20]. 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

## References

1. Alliance, F.: Client to authenticator protocol (CTAP) – proposed standard (June 2022), `https://fidoalliance.org/specs/fido-v2.1-ps-20210615/fido-client-to-authenticator-protocol-v2.1-ps-errata-20220621.html`
2. Barbosa, M., Boldyreva, A., Chen, S., Warinschi, B.: Provable security analysis of FIDO2. In: Advances in Cryptology - CRYPTO 2021. Lecture Notes in Computer Science, vol. 12827, pp. 125–156. Springer (2021). `https://doi.org/10.1007/978-3-030-84252-9_5`, `https://doi.org/10.1007/978-3-030-84252-9_5`, Full version: `https://eprint.iacr.org/2020/756`
3. Barbosa, M., Cirne, A., Esquível, L.: 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. pp. 14:1–14:11. ACM (2023). `https://doi.org/10.1145/3600160.3600174`, `https://doi.org/10.1145/3600160.3600174`
4. Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: CRYPTO 1996. pp. 1–15. Springer (1996)
5. Bindel, N., Cremers, C., Zhao, M.: Fido2, ctap 2.1, and webauthn 2: Provable security and post-quantum instantiation. In: 2023 IEEE Symposium on Security and Privacy (SP). pp. 1471–1490. IEEE (2023)
6. Bindel, N., Gama, N., Guasch, S., Ronen, E.: To attest or not to attest, this is the question–provable attestation in fido2. In: ASIACRYPT 2023. pp. 297–328. Springer (2023)
7. Canvel, B., Hiltgen, A.P., Vaudenay, S., Vuagnoux, M.: Password interception in a SSL/TLS channel. In: Advances in Cryptology - CRYPTO 2003. Lecture Notes in Computer Science, vol. 2729, pp. 583–599. Springer (2003). `https://doi.org/10.1007/978-3-540-45146-4_34`, `https://iacr.org/archive/crypto2003/27290581/27290581.pdf`
8. Dumitru, R., Genkin, D., Wabnitz, A., Yarom, Y.: The impostor among US(B): Off-Path injection attacks on USB communications. In: 32nd USENIX Security Symposium (USENIX Security 23). pp. 5863–5880. USENIX Association, Anaheim, CA (Aug 2023), `https://www.usenix.org/conference/usenixsecurity23/presentation/dumitru`
9. Dworkin, M.: Recommendation for block cipher modes of operation. Methods and Techniques (2001)
10. Esquível, L.: authenticator-rs-fork. `https://github.com/esquivel71/authenticator-rs_fork` (2025)
11. Esquível, L.: nitrokey-3-firmware-fork. `https://github.com/esquivel71/nitrokey-3-firmware_fork` (2025)

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

13. FIDO Alliance: User authentication specifications overview. `https://fidoalliance.org/specifications/` (Accessed August 2024)

14. Fischlin, M., Mittelbach, A.: An overview of the hybrid argument. Cryptology ePrint Archive, Report 2021/088 (2021), `https://eprint.iacr.org/2021/088`

15. Guirat, I.B., Halpin, H.: Formal verification of the W3C web authentication protocol. In: 5th Annual Symposium and Bootcamp on Hot Topics in the Science of Security. p. 6. ACM (2018)

16. Hanzlik, L., Loss, J., Wagner, B.: Token meets wallet: Formalizing privacy and revocation for fido2. In: 2023 IEEE Symposium on Security and Privacy (SP). pp. 1491–1508. IEEE (2023)

17. Igoe, K., McGrew, D., Salter, M.: Fundamental Elliptic Curve Cryptography Algorithms. RFC 6090 (Feb 2011). `https://doi.org/10.17487/RFC6090`, `https://www.rfc-editor.org/info/rfc6090`

18. Kepkowski, M., Hanzlik, L., Wood, I.D., Kaafar, M.A.: How not to handle keys: Timing attacks on fido authenticator privacy. Proceedings on Privacy Enhancing Technologies (2022)

19. Krawczyk, D.H., Eronen, P.: HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869 (May 2010). `https://doi.org/10.17487/RFC5869`, `https://www.rfc-editor.org/info/rfc5869`

20. Mozilla: authenticator-rs. `https://github.com/mozilla/authenticator-rs` (2025)

21. Nitrokey: nitrokey-3-firmware. `https://github.com/Nitrokey/nitrokey-3-firmware` (2025)

22. Shoup, V.: Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Paper 2004/332 (2004), `https://eprint.iacr.org/2004/332`

23. W3C: Web authentication: An API for accessing public key credentials level 2 – W3C recommendation (April 2021), `https://www.w3.org/TR/2021/REC-webauthn-2-20210408/`

24. W3C: Web authentication: An API for accessing public key credentials level 3 – W3C recommendation (September 2023), `https://www.w3.org/TR/webauthn-3/`

# A  Description of WebAuthn

This section describes the protocol algorithms of WebAuthn, as shown in Fig. 9, which follows the descriptions presented in [5,6]. 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 Fig. 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 $\hat{id}_s$, 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 $\hat{id}_s$. 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 $\mathsf{UV}$ into a message $m$ and, depending on the attestation mode required by relying party ($\mathsf{Basic}$ in Figure. 2), produces a new attestation signature $\sigma_{\mathsf{att}}$ on $m$ and $h$ (the hash of $m_{\mathsf{rcl}}$) using the authenticator's private attestation $\mathsf{ak}$. The authenticator then sets, in its registration context $\mathsf{rc}_T$, and for the server identity $\mathsf{id}_S$, the received user identifier $\mathsf{uid}$, the sampled $\mathsf{cid}$, the assertion secret key $sk$ and the signature counter $n$, and also sets the $\mathsf{id}_S$, $h$, $\mathsf{cid}$, $n$, $pk$, $\mathsf{UV}$ and attestation mode used as the agreed content $\mathsf{agCon}$ (data that must be the same both from the perspective of the server and the token). Finally, it sets the hash of $\mathsf{id}_S$, the $\mathsf{cid}$ and the signature counter as the session identifier $\mathsf{sid}$, and returns the token response message $m_{\mathsf{rrsp}}$, composed by $m$ and (optionally) $\sigma_{\mathsf{att}}$ back to the client, which in turn forwards $m_{\mathsf{rrsp}}$ and its own $m_{\mathsf{rcl}}$ to the server. To conclude the **registration**, the server runs $\mathsf{rVrfy}$, which inputs $\mathsf{id}_S$, $m_{\mathsf{rcl}}$, $m_{\mathsf{rrsp}}$ and public parameters $\mathsf{gpars}$, and then checks if the information that came from the token and client is correct, including the server identity $\mathsf{id}_S$, the token binding state $\mathsf{tb}_S$, the sampled challenge $\mathsf{ch}_S$, the user verification condition $\mathsf{UV}_S$, the signature counter $n$ (which must be zero) and, depending on the attestation mode, the attestation signature $\sigma_{\mathsf{att}}$. If everything is correct, it sets the agreed content $\mathsf{agCon}$ and the session identifier $\mathsf{id}_S$ identically to the token, sets the user identifier $\mathsf{uid}_S$, the token assertion public key $pk$ and the public signature counter $n$ in its registration context $\mathsf{rc}_S$, for the specified $\mathsf{cid}$, and finishes by returns successfully.

---

$\mathsf{rChal}(\mathsf{id}_S, \mathsf{tb}, \mathsf{UV})$:
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: $\quad\llcorner$ 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** $\mathsf{ch}_S \neq \mathsf{ch} \vee \mathsf{tb}_S \neq \mathsf{tb} \vee h \neq \mathsf{H}(\mathsf{id}_S) \vee \mathsf{UV}_S \neq \mathsf{UV} \vee n \neq 0 \vee$ $\mathsf{Ver}(\mathsf{gpars}.\mathsf{vk}, (m, \mathsf{H}(\mathsf{id}_S)), \sigma_{\mathsf{att}}) = 0$ **then**
4: $\quad\llcorner$ **return** $(0, \mathsf{rc}_S, \bot, \bot, \bot)$
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: $\quad\llcorner$ 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: $\quad\llcorner$ **return** $(0, \mathsf{rc}_S, \bot, \bot, \bot)$
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** $(1, \mathsf{rc}_S, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon})$

**Fig. 9.** WebAuthn protocol functions.

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 $\mathsf{uid}_S$, which is therefore excluded from the output server message $m_{\mathsf{ach}}$. The same is true of the aCom algorithm, which differs only in the output message $m_{\mathsf{acom}}$ that no longer contains uid. The token, upon receiving $m_{\mathsf{acom}}$, runs aRsp, which first increments the signature counter $n$ stored in its registration context. Next, the hashed server identity $\mathsf{id}_S$, along with $n$, the user verification condition UV and the hashed $m_{\mathsf{acl}}$ client message $h$ are signed using the assertion secret key $sk$ generated in the previous registration run to produce a signature $\sigma$. Then, it sets the $\mathsf{id}_S$, $h$, $n$ and UV as the agreed content agCon and the hashed $\mathsf{id}_S$, 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 $\sigma$ and the user identifier uid as the response message $m_{\mathsf{arsp}}$, which is returned to the client, and then forwarded, along with the client message $m_{\mathsf{acl}}$, to the server. The aVrfy algorithm is also similar to rVrfy. It inputs $m_{\mathsf{acl}}$ and $m_{\mathsf{arsp}}$ and verifies if the information received is correct, with the major difference being that it now verifies the assertion signature $\sigma$ 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 $\mathsf{st}_T$ 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 $\mathsf{st}_T.\mathsf{initialData}$ 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 $\mathsf{st}_T.\mathsf{version}$ and $\mathsf{st}_T.\mathsf{puvProtocolList}$.

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 $\mathsf{st}_T$. The token's retry counter $\mathsf{st}_T.\mathsf{pinRetries}$ 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 $\text{st}_T$.pinRetries is not 0, runs decapsulate to obtain $K$, decrements $\text{st}_T$.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 $\text{st}_T.m$ and, if $\text{st}_T.m$ 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 $\pi_C^j$.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 $\pi_T^i$.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 [5], 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 $\text{st}_T$ is composed of: (i) a token version $\text{st}_T$.version (e.g., CTAP2.0 or CTAP2.1), (ii) a list of available PIN/UV protocols $\text{st}_T$.puvProtocolList, (iii) the currently selected PIN/UV protocol $\text{st}_T$.puvProtocol, (iv) the counter for the maximum amount of pin failed attempts $\text{st}_T$.pinRetries, (v) the counter for the number of consecutive pin failed attempts $\text{st}_T.m$ and (vi) the stored hashed pin $\text{st}_T$.pinHash. Both tokens and clients (i.e. all session oracles $\pi_C^j$ and $\pi_T^i$) 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 of Bind. Client sessions have: (i) the selected PIN/UV protocol $\pi_C^j$.puvProtocol, and (ii) the ephemeral session key derived from ECDH $\pi_C^j$.K. Additionally, we added a new $\pi_T^i$.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).[12] 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 $2\lambda$-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. The encrypt and decrypt functions both input a $4\lambda$-bit key $K$

---

[12] 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.

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

**getPublicKey ():**
1: **return** $pk$

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

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

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

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

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

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

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

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

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

**Fig. 10.** Pin UV Auth Protocol 2. All original functions are presented as shown in [5], with our proposed addition of expand for CTAP 2.1+ in blue.

**encrypt** $(K, m)$:
1: $c \leftarrow \mathsf{SKE.Enc}(K, m)$
2: **return** $c$

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

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

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

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

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

**verify** $(K', m, t)$:
1: $t \leftarrow \mathsf{MAC}(K', m)$
2: **return** $[[t = t']]$

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

**authPowerUp-T($st_T$):**

1: **if** $st_T$.version $=\bot \wedge st_T$.puvProtocolList $=\bot$ **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($\pi_C^j$, info):**

1: Parse(version, puvProtocolList) $\leftarrow$ info
2: **if** version $= 2.0$ **then return** $\bot$
3: select puvProtocol $\leftarrow$ puvProtocolList
4: $\pi_C^j$.puvProtocol $\leftarrow$ puvProtocol
5: $\pi_C^j$.puvProtocol.initialize()
6: $\pi_C^j$.st$_{\sf exe} \leftarrow$ waiting
7: $\pi_C^j$.sid $\leftarrow \pi_C^j$.sid $||$ info $||$ puvProtocol
8: **return** puvProtocol

**obtainPinUvAuthToken-C-start($\pi_C^j$, pin):**

1: pinHash $\leftarrow$ H(pin)
2: $c_{ph} \xleftarrow{\$} \pi_C^j$.puvProtocol.encrypt($\pi_C^j$.K, pinHash)
3: $\pi_C^j$.st$_{\sf exe} \leftarrow$ bindStart
4: $\pi_C^j$.sid $\leftarrow \pi_C^j$.sid $|| c_{ph}$
5: **return** $c_{ph}$

**obtainPinUvAuthToken-C-end($\pi_C^j$, $c_{pt}$):**

1: $\pi_C^j$.bs $\leftarrow \pi_C^j$.puvProtocol.decrypt($\pi_C^j$.K, $c_{pt}$)
2: $\pi_C^j$.st$_{\sf exe} \leftarrow$ bindDone
3: $\pi_C^j$.sid $\leftarrow \pi_C^j$.sid $|| c_{pt} ||$ false
4: **return**

**setPIN-C($\pi_C^j$, pin$_U$):**

1: **if** pin$_U \notin \mathcal{P}$ **then return** $\bot$
2: $c_p \xleftarrow{\$} \pi_C^j$.puvProtocol.encrypt($\pi_C^j$.K, pin$_U$)
3: $t_p \xleftarrow{\$} \pi_C^j$.puvProtocol.authenticate($\pi_C^j$.K, $c_p$)
4: **return** $(c_p, t_p)$

**setPIN-T($\pi_T^i$, puvProtocol, $c, c_p, t_p$):**

1: **if** puvProtocol $\notin st_T$.puvProtocolList $\vee st_T$.pinHash $\neq\bot$ **then**
2:    **return** $\bot$
3: $K \leftarrow st_T$.puvProtocol.decapsulate($c$)
4: **if** $K =\bot \vee st_T$.puvProtocol.verify($K, c_p, t_p$) = false **then**
5:    **return** $\bot$
6: pin $\leftarrow st_T$.puvProtocol.decrypt($K, c_p$)
7: **if** pin $\notin \mathcal{P}$ **then return** $\bot$
8: $st_T$.pinHash $\leftarrow$ H(pin)
9: $st_T$.pinRetries $\leftarrow$ pinRetriesMax
10: **return** accepted

<span style="color:red">**auth-C ($\pi_C^j$, $M$):**</span>

1: $t \xleftarrow{\$} \pi_C^j$.puvProtocol.authenticate($\pi_C^j$.bs, $M$)
2: **return** $(M, t)$

— CTAP 2.1+ below

**auth-C ($\pi_C^j$, $M$):**

1: $(K_{\sf authC}, \_) \leftarrow \pi_C^j$.puvProtocol.expand($\pi_C^j$.bs)
2: $t \xleftarrow{\$} \pi_C^j$.puvProtocol.authenticate($K_{\sf authC}, M$)
3: **return** $(M, t)$

<span style="color:blue">**validate-C ($\pi_C^j$, $M, t$):**</span>

1: $(\_, K_{\sf authT}) \leftarrow \pi_C^j$.puvProtocol.expand($\pi_C^j$.bs)
2: **if** $\pi_C^j$.puvProtocol.verify($K_{\sf authT}, M, t$) = true **then return** accepted
3: **return** rejected

---

**getInfo-T($\pi_T^i$):**

1: info $\leftarrow (st_T$.version, $st_T$.puvProtocolList)
2: $\pi_T^i$.sid $\leftarrow \pi_T^i$.sid $||$ info

**obtainSharedSecret-T($\pi_T^i$, puvProtocol):**

1: **if** puvProtocol $\notin st_T$.puvProtocolList **then return** $\bot$
2: <span style="color:orange">$st_T$.puvProtocol.regenerate()</span>
3: $pk_T \leftarrow st_T$.puvProtocol.getPublicKey()
4: $\pi_T^i$.st$_{\sf exe} \leftarrow$ waiting
5: $\pi_T^i$.sid $\leftarrow \pi_T^i$.sid $||$ puvProtocol $|| pk_T$
6: **return** $pk_T$

**obtainSharedSecret-C-end($\pi_C^j$, $pk_T$):**

1: $(c, K) \leftarrow \pi_C^j$.puvProtocol.encapsulate($pk_T$)
2: $\pi_C^j$.K $\leftarrow K$
3: $\pi_C^j$.sid $\leftarrow \pi_C^j$.sid $|| pk_T ||$ puvProtocol $|| c$
4: **return** $c$

**obtainPinUvAuthToken-T($\pi_T^i$, puvProtocol, $c, c_{ph}$):**

1: **if** puvProtocol $\notin st_T$.puvProtocolList $\vee st_T$.pinRetries $= 0$ **then**
2:    **return** $(\bot,$ false$)$
3: $K \leftarrow st_T$.puvProtocol.decapsulate($c$)
4: **if** $K =\bot$ **then return** $(\bot,$ 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** $(\bot,$ true$)$
13:    **else**
14:      **return** $(\bot,$ 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: $\pi_T^i$.bs $\leftarrow \pi_T^i$.puvProtocol.pt
20: $c_{pt} \xleftarrow{\$} st_T$.puvProtocol.encrypt($K, \pi_T^i$.bs)
21: $\pi_T^i$.st$_{\sf exe} \leftarrow$ bindDone
22: <span style="color:blue">$\pi_T^i$.canValidate $\leftarrow$ true</span>
23: $\pi_T^i$.sid $\leftarrow \pi_T^i$.sid $||$ puvProtocol $|| c || c_{ph} || c_{pt} ||$ false
24: **return** $(c_{pt},$ false$)$

<span style="color:red">**validate-T ($\pi_T^i$, $M, t, d$):**</span>

1: **if** $st_T$.puvProtocol.verify($\pi_T^i$.bs, $M, t$) = true **then**
2:    **return** $d$
3: **return** rejected

— CTAP 2.1+ below

<span style="color:blue">**auth-T ($\pi_T^i$, $M$):**</span>

1: $(\_, K_{\sf authT}) \leftarrow \pi_C^j$.puvProtocol.expand($\pi_T^i$.bs)
2: $t \xleftarrow{\$} st_T$.puvProtocol.authenticate($K_{\sf authT}, M$)
3: **return** $(M, t)$

**validate-T ($\pi_T^i$, $M, t, d$):**

1: $(K_{\sf authC}, \_) \leftarrow \pi_C^j$.puvProtocol.expand($\pi_T^i$.bs)
2: **if** $\pi_T^i$.canValidate = true **then**
3:    **if** $st_T$.puvProtocol.verify($K_{\sf authC}, M, t$) = true **then**
4:      $\pi_T^i$.canValidate $\leftarrow$ false
5:      **return** $d$
6: **return** rejected

**Fig. 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.

and a message $m$ or ciphertext $c$, then take the last $2\lambda$ 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 $2\lambda$ or $4\lambda$ 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 $2\lambda$ 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, get-PublicKey and regenerate functions. The resetPuvToken is almost identical, but can sample a fresh pinToken that is $\lambda$-bit *or* $2\lambda$-bit in length. encrypt and decrypt receive a $2\lambda$-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 $\lambda$-bit tag $t$. In practice, $t$ is the result of truncating the leftmost $\lambda$ bits from the HMAC-SHA-256 $2\lambda$-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, $2\lambda$-bit symmetric key $K$.

**Remark.** While we generally maintained the CTAP 2.1 description as close as possible to the original work in [5], we do propose another modification to the description, which we also include in CTAP 2.1+. We include a new flag $\pi_T^i$.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 $\pi_T^i$. 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 $2\lambda$-bit pinToken $pt$ (binding state), produces a $4\lambda$-bit expanded pinToken $pt_e$ with a key derivation function modeled by $\mathcal{H}_3$ and then parses $pt_e$ into two $2\lambda$-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 $\pi_C^j$.bs to produce a new authentication key

$K_{\mathsf{authC}}$, and uses $K_{\mathsf{authC}}$ 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 $K_{\mathsf{authC}}$ and verifies $(M, t)$. After validating $(M, t)$ the token runs auth-T, which expands the binding state $\pi_T^i.\mathsf{bs}$ into a new authentication key $K_{\mathsf{authT}}$ (different from $K_{\mathsf{authC}}$) 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 $K_{\mathsf{authT}}$ 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 $H : \mathcal{K} \times \mathcal{D} \rightarrow \mathcal{R}$ 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.*

$$
\begin{aligned}
\mathsf{Adv}_H^{\mathsf{coll}}(\mathcal{A}) = \Pr[k \xleftarrow{\$} \mathcal{K}, (m_1, m_2) \xleftarrow{\$} \mathcal{A}(k) : m_1 \neq m_2 \\
\wedge\ H_k(m_1) = H_k(m_2)]\ .
\end{aligned}
$$

**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 $\mathcal{O}_a$ 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.*

$$
\mathsf{Adv}_{\mathbb{G}}^{\mathsf{sCDH}}(\mathcal{A}) = \Pr[g^{ab} \leftarrow \mathcal{A}(\mathbb{G}, g, g^a, g^b, \mathcal{O}_a)]\ .
$$

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

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

---

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

$\underline{\mathcal{O}_{\mathsf{Auth}}(m):}$
  $t \leftarrow \mathsf{MAC}(K, m)$
  $S \leftarrow S \cup (m, t)$
  **return** $t$

$\underline{\mathcal{O}_{\mathsf{Ver}}(m, t):}$
  $t' \leftarrow \mathsf{MAC}(K, m)$
  **return** $t' = t$

---

**Fig. 13.** SUF-CMA Challenger and corresponding oracles $\mathcal{O}_{\mathsf{Auth}}$ and $\mathcal{O}_{\mathsf{Ver}}$.

**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 pseudorandom permutation if, for all efficient adversary $\mathcal{A}$ and all $k \in \{0,1\}^l$, its advantage defined below is negligible.*

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

**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 $\mathsf{Expt}_{SKE}^{IND\text{-}1\$PA}$ (Fig. 14), the advantage of $\mathcal{A}$ defined below is negligible.*

$$\mathsf{Adv}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA}}(\mathcal{A}) = \Big| \ 2 \Pr[\mathsf{Expt}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA}}(\mathcal{A}) = 1] - 1 \ \Big| \ .$$

**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 $\mathsf{Expt}_{SKE}^{IND\text{-}1\$PA\text{-}LPC}$ (Fig. 15), the advantage of $\mathcal{A}$ defined below is negligible.*

$$\mathsf{Adv}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LPC}}(\mathcal{A}) = \Big| \ 2 \Pr[\mathsf{Expt}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LPC}}(\mathcal{A}) = 1] - 1 \ \Big| \ .$$

**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 $\mathsf{Expt}_{SKE}^{IND\text{-}1\$PA\text{-}LHPC}$ (Fig. 16), the advantage of $\mathcal{A}$ defined below is negligible.*

$$\mathsf{Adv}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LHPC}}(\mathcal{A}) = \Big| \ 2 \Pr[\mathsf{Expt}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LHPC}}(\mathcal{A}) = 1] - 1 \ \Big| \ .$$

**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 $\mathsf{Expt}_{SKE}^{IND\text{-}1\$PA\text{-}LHPC\text{-}H}$ (Fig. 17), the advantage of $\mathcal{A}$ defined below is negligible.*

$$\mathsf{Adv}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LHPC\text{-}H}}(\mathcal{A}) = \Big| \ 2 \Pr[\mathsf{Expt}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LHPC\text{-}H}}(\mathcal{A}) = 1] - 1 \ \Big| \ .$$

---

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

$\mathcal{O}_{\mathsf{LR}}(m_0, m_1)$:
    **if** $\mathsf{state} \neq 0$ **then**
      $\llcorner$ **return** $\bot$
    **if** $|\, m_0\,| \neq |\, m_1\,|$ **then**
      $\llcorner$ **return** $\bot$
    $c_{\mathsf{chal}} \xleftarrow{\$} \mathsf{SKE.Enc}(K, m_b)$
    $\mathsf{state} \leftarrow 1$
    **return** $c_{\mathsf{chal}}$

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

**Fig. 14.** IND-1\$PA Challenger, and corresponding Left-Right oracle $\mathcal{O}_{\mathsf{LR}}$ and RAND oracle $\mathcal{O}_{\mathsf{RAND}}$.

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

$\mathcal{O}_{\mathsf{RAND}}(l)$:
    $m'_0, m'_1 \xleftarrow{\$} \{0,1\}^l$
    $c' \xleftarrow{\$} \mathsf{SKE.Enc}(K, m'_b)$
    **return** $(m'_0, m'_1, c')$

$\mathcal{O}_{\mathsf{LR}}(m_0, m_1)$:
    **if** $\mathsf{state} \neq 0$ **then**
        **return** $\bot$
    **if** $|m_0| \neq |m_1|$ **then**
        **return** $\bot$
    $c_{\mathsf{chal}} \xleftarrow{\$} \mathsf{SKE.Enc}(K, m_b)$
    $\mathsf{state} \leftarrow 1$
    **return** $c_{\mathsf{chal}}$

$\mathcal{O}_{\mathsf{LPC}}(c)$:
    **if** $c = c_{\mathsf{chal}}$ **then**
        **return** $\bot$
    **return** $m_0 = \mathsf{SKE.Dec}(K, c)$

**Fig. 15.** IND-1\$PA-LPC Challenger, and corresponding Left-Right oracle $\mathcal{O}_{\mathsf{LR}}$, RAND oracle $\mathcal{O}_{\mathsf{RAND}}$ and LPC oracle $\mathcal{O}_{\mathsf{LPC}}$.

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

$\mathcal{O}_{\mathsf{RAND}}(l)$:
    $m'_0, m'_1 \xleftarrow{\$} \{0,1\}^l$
    $c' \xleftarrow{\$} \mathsf{SKE.Enc}(K, m'_b)$
    **return** $(m'_0, m'_1, c')$

$\mathcal{O}_{\mathsf{LR}}(m_0, m_1)$:
    **if** $\mathsf{state} \neq 0$ **then**
        **return** $\bot$
    **if** $|m_0| \neq |m_1|$ **then**
        **return** $\bot$
    $c_{\mathsf{chal}} \xleftarrow{\$} \mathsf{SKE.Enc}(K, m_b)$
    $\mathsf{state} \leftarrow 1$
    **return** $c_{\mathsf{chal}}$

$\mathcal{O}_{\mathsf{LHPC}}(c)$:
    **if** $c = c_{\mathsf{chal}}$ **then**
        **return** $\bot$
    **return** $\mathsf{H}(m_0) = \mathsf{SKE.Dec}(K, c)$

**Fig. 16.** IND-1\$PA-LHPC Challenger, and corresponding Left-Right oracle $\mathcal{O}_{\mathsf{LR}}$, RAND oracle $\mathcal{O}_{\mathsf{RAND}}$ and LHPC oracle $\mathcal{O}_{\mathsf{LHPC}}$.

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

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

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

$\mathcal{O}_{\mathsf{LHPC}}(c)$:
    $\textbf{if } c = c_{\mathsf{chal}} \textbf{ then}$
       $\llcorner \textbf{ return } \perp$
    $\textbf{return } \mathsf{H}(m_0) = \mathsf{SKE.Dec}(K, c)$

**Fig. 17.** IND-1\$PA-LHPC-H Challenger, and corresponding Left-Right oracle $\mathcal{O}_{\mathsf{LR}}$, RAND oracle $\mathcal{O}_{\mathsf{RAND}}$ and LHPC oracle $\mathcal{O}_{\mathsf{LHPC}}$.

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

# D  Security Proofs

Our proofs follow the *game-based technique* (see [22] for a tutorial) and some proofs require a *hybrid argument* (see [14] 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 $\mathsf{Pr}_i$ denote the probability that GAME $i$ outputs 1.

GAME 0. This is the original experiment. Therefore,

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

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 $\pi_C^j$ and token session $\pi_T^i$ 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, $\pi_C^j$ and $\pi_T^i$ 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 $|\mathsf{Pr}_0 - \mathsf{Pr}_1| \leq (q_S + q_E + q_{\mathsf{Send}})\,\mathsf{Adv}_{\mathsf{ECDH}}^{\mathsf{sCDH}}(\mathcal{B}_1)$.

GAME 2. In this game, in every **Setup** query, the pin is no longer stored in the token state $\mathsf{st}_T$, 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 $|\mathsf{Pr}_2 - \mathsf{Pr}_3| \leq (q_S + q_E + q_{\mathsf{NT}} + q_R + 2q_{\mathsf{Send}})^2 \,/\, (2q)$.

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

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 $|\mathsf{Pr}_4 - \mathsf{Pr}_5| \leq q_S\,\mathsf{Adv}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LHPC}}(\mathcal{B}_5)$.

GAME 6. This game replaces every encrypted hashed pin $\mathsf{H}(\mathsf{pin})$ and pinToken $pt$ in **Execute** queries with an encrypted hashed constant $\mathsf{H}(0000)$ 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 $|\mathsf{Pr}_5 - \mathsf{Pr}_6| \leq q_E\,\mathsf{Adv}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LPC}}(\mathcal{B}_6)$.

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 $\mathscr{D}$, 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 $q_{\mathsf{Send}}^{\mathsf{act}}$ the maximum number of active queries to **Send-Bind-T**, where $\mathcal{A}$ actively attempts at guessing the pin, and $1/2^{h_{\mathscr{D}}}$ as the maximum probability that $\mathcal{A}$ can guess the pin in any query, we have $|\mathsf{Pr}_7 - \mathsf{Pr}_8| \leq q_{\mathsf{Send}}^{\mathsf{act}}/2^{h_{\mathscr{D}}}$.

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 $|\mathsf{Pr}_8 - \mathsf{Pr}_9| \leq (q_E + q_{\mathsf{Send}})^2 \,/\, 2^{2\lambda+1}$.

GAME 10. In this game, when $\mathcal{A}$ queries **Validate-T** with message $M$ and a tag $t$ on a token session $\pi_T^i$ 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 $\pi_T^i$ 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 $\mathcal{B}_{10}$ against the SUF-CMA security of MAC such that if bad happens, $\mathcal{B}_{10}$ wins the SUF-CMA game. Therefore, we have $|\mathsf{Pr}_9 - \mathsf{Pr}_{10}| \leq (q_\mathsf{E} + q_\mathsf{Send}) \, \mathsf{Adv}_\mathsf{MAC}^\mathsf{SUF\text{-}CMA}(\mathcal{B}_{10})$.

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 $\mathcal{A}$. $\qquad\qquad\square$

**Remark.** In GAME 8, we bound $\mathcal{A}$'s advantage as a function of $q_\mathsf{Send}^\mathsf{act}$, 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 $q_\mathsf{NT}^\mathsf{act}$ as the number of tokens created that are actively attacked by $\mathcal{A}$ during the experiment, we could write the bound as $|\mathsf{Pr}_7 - \mathsf{Pr}_8| \leq \mathsf{pinRetriesMax} \cdot q_\mathsf{NT}^\mathsf{act}/2^{h_\mathscr{D}}$.

## 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 $|\mathsf{Pr}_9 - \mathsf{Pr}_{10}| \leq q_{\mathcal{H}_3} (q_\mathsf{E} + q_\mathsf{Send})/2^{2\lambda}$. 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 $\pi_T^i$ 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 $\pi_T^i$ 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 $|\mathsf{Pr}_{10} - \mathsf{Pr}_{11}| \leq (q_E + q_{\mathsf{Send}}) \, \mathsf{Adv}_{\mathsf{MAC}}^{\mathsf{SUF\text{-}CMA}}(\mathcal{B}_{11})$.

GAME 12. In this game, when $\mathcal{A}$ queries Validate-C with message $M$ and a tag $t$ and a client session $\pi_C^j$ 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 $|\mathsf{Pr}_{11} - \mathsf{Pr}_{12}| \leq q_E \, \mathsf{Adv}_{\mathsf{MAC}}^{\mathsf{SUF\text{-}CMA}}(\mathcal{B}_{12})$.

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 $\mathsf{Pr}_{12} = 0$. 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 $q_S$, $q_E$, $q_{\mathsf{Send}}$, $q_{\mathsf{NT}}$ 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 $\mathsf{Pr}_i$ denote the probability that $\mathbf{G}_i$ outputs 1)

$$\mathsf{Pr}_0 = \mathsf{Adv}_{\mathsf{CTAP}\ 2.1++}^{\mathsf{priv}}(\mathcal{A})/2 + 1/2 \tag{1}$$

$$\begin{aligned}
\mathrm{Pr}_0 - \mathrm{Pr}_8 \leq {} & (q_{\mathsf{S}} + q_{\mathsf{E}} + q_{\mathsf{Send}})\ \mathsf{Adv}_{\mathsf{ECDH}}^{\mathsf{sCDH}}(\mathcal{B}_1) \\
& + (q_{\mathsf{S}} + q_{\mathsf{E}} + q_{\mathsf{NT}} + q_{\mathsf{R}} + 2q_{\mathsf{Send}})^2 \ / \ (2q) \\
& + \mathsf{Adv}_{\mathsf{H}}^{\mathsf{coll}}(\mathcal{B}_4) \\
& + q_{\mathsf{S}}\ \mathsf{Adv}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LHPC}}(\mathcal{B}_5) \\
& + q_{\mathsf{E}}\ \mathsf{Adv}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LPC}}(\mathcal{B}_6) \\
& + q_{\mathsf{Send}}^{\mathsf{act}}/2^{h_{\mathscr{D}}}
\end{aligned} \tag{2}$$

$$\mathrm{Pr}_8 = 1/2 \tag{3}$$

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). Diffie-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 $\mathrm{Pr}_i$ 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 $(C, j)$, $(C, j, m_{\mathsf{acom}}, t_{\mathsf{cl}}) \notin \mathcal{L}_{\mathsf{authC}}$, which means message, tag pair $(m_{\mathsf{acom}}, t_{\mathsf{cl}})$ is not output by client session $\pi_C^j$.

Now, $\mathrm{Pr}_0 - \mathrm{Pr}_1 = \mathrm{Pr}[\mathsf{bad} \leftarrow \mathsf{true}]$. We claim that there exists a mPACA authentication adversary $\mathcal{B}_1$ such that $\mathrm{Pr}[\mathsf{bad} \leftarrow \mathsf{true}] \leq \mathsf{Adv}_{\mathsf{mPACA}}^{\mathsf{SUF\text{-}t}}(\mathcal{B}_1)$. $\mathcal{B}_1$ samples bit $b$, and simulates mPACA oracles using its own oracles provided in mPACA authentication experiment. $\mathcal{B}_1$ then internally initializes PIA 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, $\Pr[\mathsf{bad} \leftarrow \mathsf{true} \leq \mathsf{Adv}_{\mathsf{mPACA}}^{\mathsf{SUF\text{-}t}}(\mathcal{B}_1)$.

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 $|\Pr_1 - \Pr_2| \leq \mathsf{Adv}_{\mathsf{PlA}}^{\mathsf{priv}}(\mathcal{B}_2)$. 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 $b_{\mathsf{PlA}}$ is 0, $\mathcal{B}_2$ is simulating Game 0; while when $b_{\mathsf{PlA}}$ is 1, $\mathcal{C}$ is simulating Game 1.

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

We claim that there exists an mPACA privacy adversary $\mathcal{B}_3$ such that $|\Pr_2 - \Pr_3| \leq \mathsf{Adv}_{\mathsf{mPACA}}^{\mathsf{priv}}(\mathcal{B}_3)$. 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 $b_{\mathsf{mPACA}}$ is 0, $\mathcal{B}$ is simulating Game 2; while when $b_{\mathsf{mPACA}}$ is 1, $\mathcal{B}$ is simulating Game 3.

FINAL ANALYSIS. The proof is concluded as follows:

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

# 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 $H(0000)$ 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 $|\mathsf{Pr}_8 - \mathsf{Pr}_9| \le (q_\mathsf{E} + q_\mathsf{Send})^2/2^{\lambda+1}$.

GAME 10. In this game, each tag $t$ corresponds to the leftmost $\lambda$ bits from the $2\lambda$-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. [5]

We show the shortcomings of the CTAP 2.1 proof in Nina *et al.* [5] 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 $\pi_C^j$. If the hash of the PIN checks out, $\pi_T^i$ 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 $\pi_C^j$. 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 $\pi_C^j$. Since AES in CBC mode is not an authenticated encryption scheme, $\pi_C^j$ always decrypts $\tilde{c}_{pt}$, recovering a mauled pinToken of the form: $\pi_C^j.\mathsf{bs} = \mathsf{pinHash} \;||\; \mathsf{IV} \oplus \mathsf{pinHash} \oplus \mathsf{AES}(\mathsf{IV} \oplus \mathsf{pinHash})$. Formally, and this is where the proof in [5] 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 [5] (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 [5] compared to [2]. 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 $\pi_C^j$ and $\pi_T^i$, 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 $\pi_T^i$ 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 $\pi_C^j$ 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 IND-1$PA-LHPC secure (defined in Appendix C), rather than only IND-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 [5,6] to define the syntax for *passwordless authentication (PlA)* protocols. Our syntax is very similar to the ePlA protocol defined in [5] and the ePlAA protocol defined in [6]; 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 $S$,[13] a token binding state $\mathsf{tb}$, and a user verification condition $\mathsf{UV} \in \{\mathsf{T}, \mathsf{F}\}$, and outputs a challenge message $m_{\mathsf{rch}}$. 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 $\mathsf{id}_S$, a challenge message $m_{\mathsf{rch}}$, and a token binding state $\mathsf{tb}$, and outputs a client message $m_{\mathsf{rcl}}$ and a command message $m_{\mathsf{rcom}}$.

$(m_{\mathsf{rrsp}}, \mathsf{rc}_T, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \xleftarrow{\$} \mathsf{rRsp}(T, m_{\mathsf{rcom}})$**:** inputs a token $T$ and a command message $m_{\mathsf{rcom}}$ and outputs a response message $m_{\mathsf{rrsp}}$, the token-side registration context $\mathsf{rc}_T$,

---

[13] 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.

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 $m_{\mathsf{rcl}}$, a response message $m_{\mathsf{rrsp}}$, 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 $\mathsf{rc}_S$, 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 $m_{\mathsf{ach}}$. 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 $\mathsf{id}_S$, a challenge message $m_{\mathsf{ach}}$, and a token binding state tb, and outputs a client message $m_{\mathsf{acl}}$ and a command message $m_{\mathsf{acom}}$.

$(m_{\mathsf{arsp}}, \mathsf{rc}_T, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \xleftarrow{\$} \mathsf{aRsp}(T, m_{\mathsf{acom}})$: inputs a token $T$, and a command message $m_{\mathsf{acom}}$, and outputs a response message $m_{\mathsf{arsp}}$, the updated token-side registration context $\mathsf{rc}_T$, 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 $m_{\mathsf{acl}}$, and a response message $m_{\mathsf{arsp}}$, and outputs a bit $b \in \{0, 1\}$ indicating whether the authentication request was accepted. It also outputs the updated server-side registration context $\mathsf{rc}_S$, a credential identifier cid, a session identifier sid, and agreed contents agCon from the perspective of the server $S$.

**Attestation modes.** Unlike [6], 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 [6] 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.

## G.2 PlA Authentication Model and WebAuthn Authentication

Again, we closely follow [5,6] to define our authentication security model for PlA protocols. Our model is very similar to the ePlA model defined in [5] and the ePlAA model defined in [6], 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 [6]. For mode Basic, and as in [6], 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 [6]; 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 $\pi_{r,S}^i$ and $\pi_{r,T}^j$ to denote their $i$-th and $j$-th registration instances, and $\pi_{a,S}^i$ and $\pi_{a,T}^j$ to denote their $i$-th and $j$-th authentication instances. The execution status of a session oracle $\pi_{ph,P}^k$ ($ph \in \{r,a\}$, $P \in \{S,T\}$), denoted by $\pi_{ph,P}^k.\mathsf{st_{exe}}$, is either of $\{\perp, \mathsf{running}, \mathsf{accepted}\}$; here $\perp$ means the session oracle is not yet initialized, in which case we simply write $\pi_{ph,P}^k = \perp$. Session identifiers $\mathsf{sid}$ and agreed contents $\mathsf{agCon}$ are specific to a session. Registration contexts $\mathsf{rc}_S$, $\mathsf{rc}_T$ 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 $\pi_{r,S}^i$ partners with a token registration session $\pi_{r,T}^j$ if and only if $\pi_{r,S}^i.\mathsf{sid} = \pi_{r,T}^j.\mathsf{sid}$. We define partnership for authentication sessions as $\pi_{a,S}^i.\mathsf{sid} = \pi_{a,T}^j.\mathsf{sid}$, and, furthermore, we require that they can be associated to unique partnered registration sessions $\pi_{r,S}^{i'}.\mathsf{sid} = \pi_{r,T}^{j'}.\mathsf{sid}$ such that $\pi_{a,S}^i.\mathsf{cid} = \pi_{r,S}^{i'}.\mathsf{cid}$. 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 [5,6], 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 [2].

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

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

**Authentication security of WebAuthn.** We do not restate the authentication security of WebAuthn in this model, as it has been proved in [2]. Indeed, the security proofs given in [2] 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 [2], 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 [5,6], in order to capture a guarantee provided by FIDO2, which was captured by the original model in [2] but lost in the more recent works [5,6]: a separation between registration and authentication sessions for modeling the partnership between them, and the explicit handling of credential identifiers and group identifiers.[14] 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 server can only accept registrations from tokens whose attestation material is consistent with a given set of group parameters.

---

[14] 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: $\quad$ Retrieve $(S, i', \mathsf{gid}, \mathsf{cid})$ from $\mathcal{L}_{\mathsf{reg}}$
3: $\quad$ **if** $\exists T, j'$ such that $\pi^{i'}_{r,S}.\mathsf{sid} = \pi^{j'}_{r,T}.\mathsf{sid}$ **then**
4: $\quad\quad$ **if** $\exists j$ such that $\pi^i_{a,S}.\mathsf{sid} = \pi^j_{a,T}.\mathsf{sid}$ **then**
5: $\quad\quad\quad$ **return** $(\mathsf{gid}, i', T, j', j)$
6: $\quad\quad$ **else return** $(\mathsf{gid}, i', T, j', \bot)$
7: $\quad$ **else return** $(\mathsf{gid}, i', \bot, \bot, \bot)$
8: **return** $(\bot, \bot, \bot, \bot, \bot)$

Win-auth $(S, i, \mathsf{cid})$:
1: **if** $\exists(S_1, i_1, ph_1) \neq (S_2, i_2, ph_2)$ s.t. $\pi^{i_1}_{ph_1,S_1}.\mathsf{sid} = \pi^{i_2}_{ph_2,S_2}.\mathsf{sid} \neq \bot$ **then return** $1$

2: **if** $\exists(T_1, j_1, ph_1) \neq (T_2, j_2, ph_2)$ s.t. $\pi^{j_1}_{ph_1,T_1}.\mathsf{sid} = \pi^{j_2}_{ph_2,T_2}.\mathsf{sid} \neq \bot$ **then return** $1$

3: **if** $\exists(S_1, i_1) \neq (S_2, i_2)$ s.t. $\pi^{i_1}_{r,S_1}.\mathsf{cid} = \pi^{i_2}_{r,S_2}.\mathsf{cid} \neq \bot$ **then**
4: $\quad$ **return** $1$

5: **if** $\exists(S', i', ph'), (T', j', ph')$ s.t. $\pi^{i'}_{ph',S'}.\mathsf{sid} = \pi^{j'}_{ph',T'}.\mathsf{sid} \neq \bot$ **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** $\mathsf{gid} = \bot$ **or** $(\mathsf{gid} \notin \mathcal{L}_{\mathsf{corrG}}$ **and** $(T = \bot$ **or** $T.\mathsf{gid} \neq \mathsf{gid}))$ **then**
9: $\quad$ **return** $1$
10: **else**
11: $\quad$ // Authentication broken: no authentication partner
12: $\quad$ **if** $\mathsf{gid} \notin \mathcal{L}_{\mathsf{corrG}}$ **then**
13: $\quad\quad$ **if** $(S, T) \notin \mathcal{L}_{\mathsf{corr}}$ **and** $j = \bot$ **then return** $1$
14: $\quad$ **return** $0$

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

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** $T.\mathsf{gid} = \bot$ **or** $\pi^i_{r,S} \neq \bot$ **or** $\pi^j_{r,T} \neq \bot$ **or** $\mathsf{rc}_T[S] \neq \bot$ **or** $G[\mathsf{gid}] = \bot$ **then return** $\bot$
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** $G[\mathsf{gid}] = \bot$ **then return** $\bot$
2: **if** $T.\mathsf{gid} \neq \bot$ **then return** $\bot$
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** $\pi^i_{r,S} \neq \bot$ **then return** $\bot$
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** $\pi^i_{r,S} = \bot$ **or** $\pi^i_{r,S}.\mathsf{st_{exe}} \neq$ running **or** $G[\mathsf{gid}] = \bot$ **then return** $\bot$
2: $(d, \mathsf{rc}_S, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \xleftarrow{\$}$ $\quad \mathsf{rVrfy}(\pi^i_{r,S}, m_{\mathsf{rcl}}, m_{\mathsf{rrsp}}, G[\mathsf{gid}].\mathsf{gpars})$
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** $\mathsf{rc}_T[S] = \bot$ **then return** $\bot$
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** $\pi^j_{r,T} \neq \bot$ **or** $T.\mathsf{gid} = \bot$ **then return** $\bot$
2: $(m_{\mathsf{rrsp}}, \mathsf{rc}_T, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \xleftarrow{\$}$ $\quad \mathsf{rRsp}(\pi^j_{r,T}, m_{\mathsf{rcom}})$
3: $\mathcal{L}^r_{\mathsf{ch}} \leftarrow \mathcal{L}^r_{\mathsf{ch}} \cup \{\mathsf{cid}\}$
4: **return** $m_{\mathsf{rrsp}}$

aChall $((S, i), \mathsf{tb}, \mathsf{UV})$:
1: **if** $\pi^i_{a,S} \neq \bot$ **then return** $\bot$
2: $m_{\mathsf{ach}} \xleftarrow{\$} \mathsf{aChal}(\pi^i_{a,S}, \mathsf{tb}, \mathsf{UV})$
3: **return** $m_{\mathsf{ach}}$

CorruptGroup $(\mathsf{gid})$:
1: **if** $G[\mathsf{gid}] = \bot$ **then return** $\bot$
2: $\mathcal{L}_{\mathsf{corrG}} \leftarrow \mathcal{L}_{\mathsf{corrG}} \cup \{\mathsf{gid}\}$
3: **return** $G[\mathsf{gid}]$

aCompl $((S, i), m_{\mathsf{acl}}, m_{\mathsf{arsp}})$:
1: **if** $\pi^i_{a,S} = \bot$ **or** $\pi^i_{a,S}.\mathsf{st_{exe}} \neq$ running **then return** $\bot$
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** $\mathsf{win\text{-}auth} = 0$ **then**
4: $\quad \mathsf{win\text{-}auth} \leftarrow$ Win-auth$(S, i, \mathsf{cid})$
5: **return** $d$

aResp $((T, j), m_{\mathsf{acom}})$:
1: **if** $\pi^j_{a,T} \neq \bot$ **or** $T.\mathsf{gid} = \bot$ **then return** $\bot$
2: $(m_{\mathsf{arsp}}, \mathsf{rc}_T, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \xleftarrow{\$}$ $\quad \mathsf{aRsp}(\pi^j_{a,T}, m_{\mathsf{acom}})$
3: $\mathcal{L}^a_{\mathsf{ch}} \leftarrow \mathcal{L}^a_{\mathsf{ch}} \cup \{\mathsf{cid}\}$
4: **return** $m_{\mathsf{arsp}}$

**Fig. 18.** Security experiment, winning conditions, and oracle definitions for PlA authentication security experiment. Code in blue represents the added modifications with respect to [5,6]. Code in teal is unique to the PlA privacy experiment as defined in Fig. 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.

## G.3 PlA Privacy Model and WebAuthn Privacy

Our privacy PlA model closely follows [6]. 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 [16] and add credential id cid to $\mathcal{L}$ instead. Compared to [6] 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. [6] 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 [6], 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, [16] 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 [6] [16]. 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

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

We remark that in [6] 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 [16] 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*

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

We note that the proofs in [6] still work for the stronger model.

**Fig. 19.** Experiment $\mathsf{Expt}^{\mathsf{priv}}_{\mathsf{PIA}}$ 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 [6].

# 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 [2,5], 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 [2,5], 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 [5], 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 [5] by using $\tilde{\pi}$ and $\pi$ 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 Fig. 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 $\tilde{\pi}_{r,S}^i$ by calling rCom to produce a client message $m_{\mathsf{rcl}}$ and a command $m_{\mathsf{rcom}}$. This command is then authenticated using the mPACA oracle Auth-C, producing a tag $t_{\mathsf{cl}}$. In addition to $m_{\mathsf{rch}}$, both messages output by rCom and $t_{\mathsf{cl}}$ are given to $\mathcal{A}$. The changes to aChall are analogous. The rResp oracle now additionally inputs an mPACA token session, a tag $t_{\mathsf{cl}}$ and a user decision bit $d$. It starts by querying the Validate-T oracle on the $m_{\mathsf{rcom}}$ message and tag $t_{\mathsf{cl}}$, aborting if the status is not accepted. Then, it authenticates the PlA token response $m_{\mathsf{rrsp}}$ via the mPACA oracle Auth-T, producing a tag $t_{\mathsf{tk}}$, which is also given to $\mathcal{A}$. The changes to aResp are analogous. Finally, the rCompl oracle additionally receives a tag $t_{\mathsf{tk}}$, fetches the client session associated with the current server session, and queries Validate-C on the token response $m_{\mathsf{rrsp}}$ and $t_{\mathsf{tk}}$, 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 $\tilde{\pi}_{a,S}^i$ 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 Fig. 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

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

**Differences to prior work.** We adopt the style of presentation of [5], but our composed model expresses the winning condition in a way that is closer to the one defined in [2]: 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

rChall $(S, i, C, k, \mathsf{tb}, UV)$:
1: **if** $\tilde{\pi}_{r,S}^i \neq \perp$ **then return** $\perp$
2: $m_{\mathsf{rch}} \xleftarrow{\$} \mathsf{rChal}(\tilde{\pi}_{r,S}^i, \mathsf{tb}, \mathsf{UV})$
3: $(m_{\mathsf{rcl}}, m_{\mathsf{rcom}}) \xleftarrow{\$} \mathsf{rCom}(\mathsf{id}_S, m_{\mathsf{rch}}, \mathsf{tb})$
4: $\mathsf{resp} \xleftarrow{\$} \mathsf{Auth\text{-}C}(C, k, m_{\mathsf{rcom}})$
5: **if** $\mathsf{resp} = \perp$ **then return** $\perp$
6: $(m_{\mathsf{rcom}}, t_{\mathsf{cl}}) \leftarrow \mathsf{resp}$
7: $\mathcal{L}_{\mathsf{pla\text{-}paca\text{-}S}} \leftarrow \mathcal{L}_{\mathsf{pla\text{-}paca\text{-}S}} \cup \{(\mathsf{reg}, S, i, C, k)\}$
8: **return** $(m_{\mathsf{rch}}, m_{\mathsf{rcl}}, m_{\mathsf{rcom}}, t_{\mathsf{cl}})$

rResp $(T, j, j', m_{\mathsf{rcom}}, t_{\mathsf{cl}}, d)$:
1: **if** $\tilde{\pi}_{r,T}^j \neq \perp$ **or** $T.\mathsf{gid} = \perp$ **then return** $\perp$
2: $\mathsf{status} \xleftarrow{\$} \mathsf{Validate\text{-}T}(T, j', m_{\mathsf{rcom}}, t_{\mathsf{cl}}, d)$
3: **if** $\mathsf{status} \neq \mathsf{accepted}$ **then return** $\perp$
4: $(m_{\mathsf{rrsp}}, \mathsf{rc}_T, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \xleftarrow{\$} \mathsf{rRsp}(\tilde{\pi}_{r,T}^j, m_{\mathsf{rcom}})$
5: $(m_{\mathsf{rrsp}}, t_{\mathsf{tk}}) \xleftarrow{\$} \mathsf{Auth\text{-}T}(T, j', m_{\mathsf{rrsp}})$
6: $\mathcal{L}_{\mathsf{pla\text{-}paca\text{-}T}} \leftarrow \mathcal{L}_{\mathsf{pla\text{-}paca\text{-}T}} \cup \{(\mathsf{reg}, T, j, j')\}$
7: **return** $(m_{\mathsf{rrsp}}, t_{\mathsf{tk}})$

rCompl $(S, i, m_{\mathsf{rcl}}, m_{\mathsf{rrsp}}, \mathsf{gid}, t_{\mathsf{tk}})$:
1: **if** $\tilde{\pi}_{r,S}^i = \perp$ **or** $\tilde{\pi}_{r,S}^i.\mathsf{st}_{\mathsf{exe}} \neq \mathsf{running}$ **or** $G[\mathsf{gid}] = \perp$ **then return** $\perp$
2: Retrieve $(\mathsf{reg}, S, i, C, k)$ from $\mathcal{L}_{\mathsf{pla\text{-}paca\text{-}S}}$
3: $\mathsf{status} \leftarrow \mathsf{Validate\text{-}C}(C, k, m_{\mathsf{rrsp}}, t_{\mathsf{tk}})$
4: **if** $\mathsf{status} \neq \mathsf{accepted}$ **then return** $\perp$
5: $(d, \mathsf{rc}_S, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \xleftarrow{\$} \mathsf{rVrfy}(\tilde{\pi}_{r,S}^i, m_{\mathsf{rcl}}, m_{\mathsf{rrsp}}, G[\mathsf{gid}].\mathsf{gpars})$
6: **if** $d = 1$ **then**
7: $\quad \mathcal{L}_{\mathsf{reg}} \leftarrow \mathcal{L}_{\mathsf{reg}} \cup \{(S, i, \mathsf{gid}, \mathsf{cid})\}$
8: **return** $d$

aChall $(S, i, C, k, \mathsf{tb}, \mathsf{UV})$:
1: **if** $\tilde{\pi}_{a,S}^i \neq \perp$ **then return** $\perp$
2: $m_{\mathsf{ach}} \xleftarrow{\$} \mathsf{aChal}(\tilde{\pi}_{a,S}^i, \mathsf{tb}, \mathsf{UV})$
3: $(m_{\mathsf{acl}}, m_{\mathsf{acom}}) \xleftarrow{\$} \mathsf{aCom}(\mathsf{id}_S, m_{\mathsf{ach}}, \mathsf{tb})$
4: $\mathsf{resp} \xleftarrow{\$} \mathsf{Auth\text{-}C}(C, k, m_{\mathsf{acom}})$
5: **if** $\mathsf{resp} = \perp$ **then return** $\perp$
6: $(m_{\mathsf{acom}}, t_{\mathsf{cl}}) \leftarrow \mathsf{resp}$
7: $\mathcal{L}_{\mathsf{pla\text{-}paca\text{-}S}} \leftarrow \mathcal{L}_{\mathsf{pla\text{-}paca\text{-}S}} \cup \{(\mathsf{auth}, S, i, C, k)\}$
8: **return** $(m_{\mathsf{ach}}, m_{\mathsf{acl}}, m_{\mathsf{acom}}, t_{\mathsf{cl}})$

aResp $(T, j, j', m_{\mathsf{acom}}, t_{\mathsf{cl}}, d)$:
1: **if** $\tilde{\pi}_{a,T}^j \neq \perp$ **or** $T.\mathsf{gid} = \perp$ **then return** $\perp$
2: $\mathsf{status} \xleftarrow{\$} \mathsf{Validate\text{-}T}(T, j', m_{\mathsf{acom}}, t_{\mathsf{cl}}, d)$
3: **if** $\mathsf{status} \neq \mathsf{accepted}$ **then return** $\perp$
4: $(m_{\mathsf{arsp}}, \mathsf{rc}_T, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \xleftarrow{\$} \mathsf{aRsp}(\tilde{\pi}_{a,T}^j, m_{\mathsf{acom}})$
5: $(m_{\mathsf{arsp}}, t_{\mathsf{tk}}) \xleftarrow{\$} \mathsf{Auth\text{-}T}(T, j', m_{\mathsf{arsp}})$
6: $\mathcal{L}_{\mathsf{pla\text{-}paca\text{-}T}} \leftarrow \mathcal{L}_{\mathsf{pla\text{-}paca\text{-}T}} \cup \{(\mathsf{auth}, T, j, j')\}$
7: **return** $(m_{\mathsf{arsp}}, t_{\mathsf{tk}})$

aCompl $(S, i, m_{\mathsf{acl}}, m_{\mathsf{arsp}}, t_{\mathsf{tk}})$:
1: **if** $\tilde{\pi}_{a,S}^i = \perp$ **or** $\tilde{\pi}_{a,S}^i.\mathsf{st}_{\mathsf{exe}} \neq \mathsf{running}$ **then return** $\perp$
2: Retrieve $(\mathsf{auth}, S, i, C, k)$ from $\mathcal{L}_{\mathsf{pla\text{-}paca\text{-}S}}$
3: $\mathsf{status} \leftarrow \mathsf{Validate\text{-}C}(C, k, m_{\mathsf{arsp}}, t_{\mathsf{tk}})$
4: **if** $\mathsf{status} \neq \mathsf{accepted}$ **then return** $\perp$
5: $(d, \mathsf{rc}_S, \mathsf{cid}, \mathsf{sid}, \mathsf{agCon}) \xleftarrow{\$} \mathsf{aVrfy}(\tilde{\pi}_{a,S}^i, m_{\mathsf{acl}}, m_{\mathsf{arsp}})$
6: **if** $d = 1$ **and** $\mathsf{win\text{-}ua} = 0$ **then**
7: $\quad \mathsf{win\text{-}ua} \leftarrow \mathsf{Win\text{-}ua}(S, i, \mathsf{cid})$
8: **return** $d$

**Fig. 20.** Oracle definitions for $\mathsf{ua}$ 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.

$\mathsf{Expt}^{\mathsf{ua}}_{\mathsf{PlA+mPACA}}(\mathcal{A})$:
1: $\mathcal{L}_{\mathsf{reg}} \leftarrow \emptyset;\ \mathcal{L}_{\mathsf{corr}} \leftarrow \emptyset;\ \mathcal{L}_{\mathsf{corrG}} \leftarrow \emptyset;\ G \leftarrow \emptyset;\ \mathsf{gid} \leftarrow 0$ // From PlA
2: $\mathcal{L}_{\mathsf{authC}}, \mathcal{L}_{\mathsf{authT}} \leftarrow \emptyset$ // From mPACA
3: $\mathcal{L}_{\mathsf{pla\text{-}paca\text{-}S}}, \mathcal{L}_{\mathsf{pla\text{-}paca\text{-}T}} \leftarrow \emptyset$ // For grouping PlA and mPACA sessions
4: $\mathsf{win\text{-}ua} \leftarrow 0$
5: $() \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(1^\lambda)$
6: **return** $\mathsf{win\text{-}ua}$

$\mathsf{Win\text{-}ua}\ (S, i, \mathsf{cid})$:
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. $\tilde{\pi}^{i_1}_{ph_1, S_1}.\mathsf{sid} = \tilde{\pi}^{i_2}_{ph_2, S_2}.\mathsf{sid} \neq \perp$ **then return** 1
3: **if** $\exists (T_1, j_1, ph_1) \neq (T_2, j_2, ph_2)$ s.t. $\tilde{\pi}^{j_1}_{ph_1, T_1}.\mathsf{sid} = \tilde{\pi}^{j_2}_{ph_2, T_2}.\mathsf{sid} \neq \perp$ **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** $\pi^{k_1}_{C_1}.\mathsf{st_{exe}} = \pi^{k_2}_{C_2}.\mathsf{st_{exe}} = \mathsf{bindDone}$
$\quad$ **and** $\pi^{k_1}_{C_1}.\mathsf{sid} = \pi^{k_2}_{C_2}.\mathsf{sid}$ **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** $\pi^{j'_1}_{T_1}.\mathsf{st_{exe}} = \pi^{j'_2}_{T_2}.\mathsf{st_{exe}} = \mathsf{bindDone}$
$\quad$ **and** $\pi^{j'_1}_{T_1}.\mathsf{sid} = \pi^{j'_2}_{T_2}.\mathsf{sid}$ **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. $\tilde{\pi}^{i_1}_{r, S_1}.\mathsf{cid} = \tilde{\pi}^{i_2}_{r, S_2}.\mathsf{cid} \neq \perp$ **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. $\tilde{\pi}^{i'}_{ph', S'}.\mathsf{sid} = \tilde{\pi}^{j'}_{ph', T'}.\mathsf{sid} \neq \perp$ **and** $(S', T') \notin \mathcal{L}_{\mathsf{corr}}$ **and** $T'.\mathsf{gid} \notin \mathcal{L}_{\mathsf{corrG}}$ **and** $\tilde{\pi}^{i'}_{ph', S'}.\mathsf{agCon} \neq \tilde{\pi}^{j'}_{ph', T'}.\mathsf{agCon}$
$\quad$ **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: $(\mathsf{gid}, i', T, j', j) \leftarrow \mathsf{Partnerships}(S, i, \mathsf{cid})$
17: **if** $\mathsf{gid} = \perp$ **or** $(\mathsf{gid} \notin \mathcal{L}_{\mathsf{corrG}}$ **and** $(T = \perp$ **or** $T.\mathsf{gid} \neq \mathsf{gid}))$ **then return** 1
18: **else**
19: $\quad$ **if** $\mathsf{gid} \notin \mathcal{L}_{\mathsf{corrG}}$ **then**
20: $\quad\quad$ // 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: $\quad\quad$ Retrieve $(\mathsf{reg}, T, j', l')$ from $\mathcal{L}_{\mathsf{pla\text{-}paca\text{-}T}}$ // 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: $\quad\quad$ Retrieve $(\mathsf{reg}, S, i', C_S, k_S)$ from $\mathcal{L}_{\mathsf{pla\text{-}paca\text{-}S}}$
23: $\quad\quad$ $(C_T, k_T) \leftarrow \mathsf{tokenBindPartner}(T, l')$
24: $\quad\quad$ // 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: $\quad\quad$ **if** $(C_S, k_S) \neq (C_T, k_T)$ **then**
26: $\quad\quad\quad$ $(T_S, m) \leftarrow \mathsf{clientBindPartner}(C_S, k_S)$
27: $\quad\quad\quad$ **if** $\pi^{k_S}_{C_S}.\mathsf{compromised} = \mathsf{false}$ **and** $\pi^m_{T_S}.\mathsf{pinCorr} = \mathsf{false}$ **then return** 1
28: $\quad\quad\quad$ **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
29: $\quad\quad$ **if** $(S, T) \notin \mathcal{L}_{\mathsf{corr}}$ **and** $j = \perp$ **then return** 1
30: $\quad\quad$ **else if** $(S, T) \notin \mathcal{L}_{\mathsf{corr}}$ **then**
31: $\quad\quad\quad$ // 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: $\quad\quad\quad$ Retrieve $(\mathsf{auth}, T, j, l)$ from $\mathcal{L}_{\mathsf{pla\text{-}paca\text{-}T}}$ // 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: $\quad\quad\quad$ Retrieve $(\mathsf{auth}, S, i, C'_S, k'_S)$ from $\mathcal{L}_{\mathsf{pla\text{-}paca\text{-}S}}$
34: $\quad\quad\quad$ $(C'_T, k'_T) \leftarrow \mathsf{tokenBindPartner}(T, l)$
35: $\quad\quad\quad$ **if** $(C'_S, k'_S) \neq (C'_T, k'_T)$ **then**
36: $\quad\quad\quad\quad$ $(T'_S, m') \leftarrow \mathsf{clientBindPartner}(C_S, k_S)$
37: $\quad\quad\quad\quad$ **if** $\pi^{k'_S}_{C'_S}.\mathsf{compromised} = \mathsf{false}$ **and** $\pi^{m'}_{T'_S}.\mathsf{pinCorr} = \mathsf{false}$ **then return** 1
38: $\quad\quad\quad\quad$ **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
39: **return** 0

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

checks.[15] We can do this because we establish a stronger result based on mPACA. A winning condition closer to the one in [5] 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 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 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

---

[15] The authors in [2] 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.

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.

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.

```
rResp (T, j, j', m_rcom, t_cl, d):                      aResp (T, j, j', m_acom, t_cl, d):
 1: if π̃_{r,T}^j ≠⊥ or T.gid =⊥ then return ⊥           1: if π̃_{a,T}^j ≠⊥ or T.gid =⊥ then return ⊥
 2: status ←$ Validate-T(T, j', m_rcom, t_cl, d)         2: status ←$ Validate-T(T, j', m_acom, t_cl, d)
 3: if status ≠ accepted then return ⊥                   3: if status ≠ accepted then return ⊥
 4: if Token-Win-SUF-t(T, j', m_rcom, t_cl, d) = 1 then  4: if Token-Win-SUF-t(T, j', m_acom, t_cl, d) = 1 then
 5: ⌊  game ends returning 0                             5: ⌊  game ends returning 0
 6: (m_rrsp, rc_T, cid, sid, agCon) ←$ rRsp(π̃_{r,T}^j, m_rcom)   6: (m_arsp, rc_T, cid, sid, agCon) ←$ aRsp(π̃_{a,T}^j, m_acom)
 7: (m_rrsp, t_tk) ←$ Auth-T(T, j', m_rrsp)              7: (m_arsp, t_tk) ←$ Auth-T(T, j', m_arsp)
 8: L_pla-paca-T ← L_pla-paca-T ∪ {(reg, T, j, j')}       8: L_pla-paca-T ← L_pla-paca-T ∪ {(auth, T, j, j')}
 9: return (m_rrsp, t_tk)                                9: return (m_arsp, t_tk)


rCompl (S, i, m_rcl, m_rrsp, gid, t_tk):                 aCompl (S, i, m_acl, m_arsp, t_tk):
 1: if π̃_{r,S}^i =⊥ or π̃_{r,S}^i.st_exe ≠ running or G[gid] =⊥ then re-   1: if π̃_{a,S}^i =⊥ or π̃_{a,S}^i.st_exe ≠ running then return ⊥
    turn ⊥                                               2: Retrieve (auth, S, i, C, k) from L_pla-paca-S
 2: Retrieve (reg, S, i, C, k) from L_pla-paca-S          3: status ← Validate-C(C, k, m_arsp, t_tk)
 3: status ← Validate-C(C, k, m_rrsp, t_tk)              4: if status ≠ accepted then return ⊥
 4: if status ≠ accepted then return ⊥                   5: if Client-Win-SUF-t(C, k, m_acom, t_tk) = 1 then
 5: if Client-Win-SUF-t(C, k, m_rcom, t_tk) = 1 then     6: ⌊  game ends returning 0
 6: ⌊  game ends returning 0                             7: (d, rc_S, cid, sid, agCon) ←$
 7: (d, rc_S, cid, sid, agCon) ←$                           aVrfy(π̃_{a,S}^i, m_acl, m_arsp)
    rVrfy(π̃_{r,S}^i, m_rcl, m_rrsp, G[gid].gpars)        8: if d = 1 and win-ua = 0 then
 8: if d = 1 then                                        9: ⌊  win-ua ← Win-ua(S, i, cid)
 9: ⌊  L_reg ← L_reg ∪ {(S, i, gid, cid)}               10: return d
10: return d
```

**Fig. 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.

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.

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.

Win-ua $(S, i, \mathsf{cid})$:

1: **if** $\exists (S_1, i_1, ph_1) \neq (S_2, i_2, ph_2)$ s.t. $\tilde{\pi}_{ph_1, S_1}^{i_1}.\mathsf{sid} = \tilde{\pi}_{ph_2, S_2}^{i_2}.\mathsf{sid} \neq \perp$ **then** game ends returning 0

2: **if** $\exists (T_1, j_1, ph_1) \neq (T_2, j_2, ph_2)$ s.t. $\tilde{\pi}_{ph_1, T_1}^{j_1}.\mathsf{sid} = \tilde{\pi}_{ph_2, T_2}^{j_2}.\mathsf{sid} \neq \perp$ **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** $\pi_{C_1}^{k_1}.\mathsf{st_{exe}} = \pi_{C_2}^{k_2}.\mathsf{st_{exe}} = \mathsf{bindDone}$
   $\quad$ **and** $\pi_{C_1}^{k_1}.\mathsf{sid} = \pi_{C_2}^{k_2}.\mathsf{sid}$ **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** $\pi_{T_1}^{j_1'}.\mathsf{st_{exe}} = \pi_{T_2}^{j_2'}.\mathsf{st_{exe}} = \mathsf{bindDone}$
   $\quad$ **and** $\pi_{T_1}^{j_1'}.\mathsf{sid} = \pi_{T_2}^{j_2'}.\mathsf{sid}$ **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** $\pi_{T_1}^{j_1'}.\mathsf{st_{exe}} = \pi_{T_2}^{j_2'}.\mathsf{st_{exe}} = \mathsf{bindDone}$
   $\quad$ **and** $\pi_{T_1}^{j_1'}.\mathsf{sid} = \pi_{T_2}^{j_2'}.\mathsf{sid}$ **then** game ends returning 0

7:

8: **if** $\exists (S_1, i_1) \neq (S_2, i_2)$ s.t. $\tilde{\pi}_{r, S_1}^{i_1}.\mathsf{cid} = \tilde{\pi}_{r, S_2}^{i_2}.\mathsf{cid} \neq \perp$ **then** game ends returning 0

9:

10: **if** $\exists (S', i', ph'), (T', j', ph')$ s.t. $\tilde{\pi}_{ph', S'}^{i'}.\mathsf{sid} = \tilde{\pi}_{ph', T'}^{j'}.\mathsf{sid} \neq \perp$ **and** $(S', T') \notin \mathcal{L}_{\mathsf{corr}}$ **and** $T'.\mathsf{gid} \notin \mathcal{L}_{\mathsf{corrG}}$ **and** $\tilde{\pi}_{ph', S'}^{i'}.\mathsf{agCon} \neq$
   $\quad \tilde{\pi}_{ph', T'}^{j'}.\mathsf{agCon}$ **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: $\quad$ **if** $\mathsf{gid} \notin \mathcal{L}_{\mathsf{corrG}}$ **then**

16: $\quad\quad$ Retrieve $(\mathsf{reg}, T, j', l')$ from $\mathcal{L}_{\mathsf{pla\text{-}paca\text{-}T}}$

17: $\quad\quad$ Retrieve $(\mathsf{reg}, S, i', C_S, k_S)$ from $\mathcal{L}_{\mathsf{pla\text{-}paca\text{-}S}}$

18: $\quad\quad$ $(C_T, k_T) \leftarrow \mathsf{tokenBindPartner}(T, l')$

19: $\quad\quad$ **if** $(C_S, k_S) \neq (C_T, k_T)$ **then**

20: $\quad\quad\quad$ $(T_S, m) \leftarrow \mathsf{clientBindPartner}(C_S, k_S)$

21: $\quad\quad\quad$ **if** $\pi_{C_S}^{k_S}.\mathsf{compromised} = \mathsf{false}$ **and** $\pi_{T_S}^{m}.\mathsf{pinCorr} = \mathsf{false}$ **then return** 1

22: $\quad\quad\quad$ **if** $((C_T, k_T) = (\perp, \perp)$ **or** $\pi_{C_T}^{k_T}.\mathsf{compromised} = \mathsf{false})$ **and** $\pi_{T}^{l'}.\mathsf{pinCorr} = \mathsf{false}$ **then return** 1

23: $\quad\quad$ **if** $(S, T) \notin \mathcal{L}_{\mathsf{corr}}$ **and** $j = \perp$ **then** game ends returning 0

24: $\quad$ **else if** $(S, T) \notin \mathcal{L}_{\mathsf{corr}}$ **then**

25: $\quad\quad$ Retrieve $(\mathsf{auth}, T, j, l)$ from $\mathcal{L}_{\mathsf{pla\text{-}paca\text{-}T}}$

26: $\quad\quad$ Retrieve $(\mathsf{auth}, S, i, C_S', k_S')$ from $\mathcal{L}_{\mathsf{pla\text{-}paca\text{-}S}}$

27: $\quad\quad$ $(C_T', k_T') \leftarrow \mathsf{tokenBindPartner}(T, l)$

28: $\quad\quad$ **if** $(C_S', k_S') \neq (C_T', k_T')$ **then**

29: $\quad\quad\quad$ $(T_S', m') \leftarrow \mathsf{clientBindPartner}(C_S, k_S)$

30: $\quad\quad\quad$ **if** $\pi_{C_S'}^{k_S'}.\mathsf{compromised} = \mathsf{false}$ **and** $\pi_{T_S'}^{m'}.\mathsf{pinCorr} = \mathsf{false}$ **then return** 1

31: $\quad\quad\quad$ **if** $((C_T', k_T') = (\perp, \perp)$ **or** $\pi_{C_T'}^{k_T'}.\mathsf{compromised} = \mathsf{false})$ **and** $\pi_{T}^{l}.\mathsf{pinCorr} = \mathsf{false}$ **then return** 1

32: **return** 0

**Fig. 23.** Second hop of the composed model proof. Code in red represents the changes to the code corresponding to the second hop.

$\square$

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

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

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 [5].

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.[16]

---

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

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 [5] 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).

---

$\underline{\mathcal{A}^{\mathsf{priv}}_{\mathsf{CTAP\ 2.1/CTAP\ 2.1+}}(\lambda)}$:

For some distinct tokens $T_0, T_1$, some clients $C_0, C_1$, and some user $U$

1: Phase 1: $\mathsf{trans}_0 \leftarrow \mathbf{Setup}(T_0, 1, C_0, 1, U)$, $\mathsf{trans}_1 \leftarrow \mathbf{Setup}(T_1, 1, C_1, 1, U)$
2: Extract token $T$'s DH share $pk_0$ from $\mathsf{trans}_0$ and DH share $pk_1$ from $\mathsf{trans}_1$
3: Phase 2: Outputs $T_0, T_1, C_0, C_1, U, U$
4: Phase 3: $\mathsf{trans} \leftarrow \mathbf{Bind\text{-}LEFT}_{T_0, C_0}(2, 2)$
5: Extract token $T$'s DH share $pk$ from $\mathsf{trans}$
6: **if** $pk = pk_0$ **then**
7: $\quad$ **return** 0
8: **else**
9: $\quad$ **return** 1

---

**Fig. 24.** Privacy adversary for CTAP 2.1 and CTAP 2.1+.

## K    IND-1\$PA-LHPC Proof for CBC

In this section, we show our code-based proof of the IND-1\$PA-LHPC security of CBC via a sequence of games. Our proof is based on the IND-1\$PA-LPC proof for CBC from [5], with

a few modifications to accomodate the differences in the two security definitions. Therefore, much like the authors in [5], we define CBC as an instance of a symmetric encryption scheme SKE = (Kg, Enc, Dec), seen in 25. We also define F as an invertible pseudo-random permutation F : $\mathcal{K} \times \mathcal{M} \to \mathcal{C}$, with $\mathcal{K} := \{0,1\}^{f_1(\lambda)}$, $\mathcal{M} := \{0,1\}^{f_2(\lambda)}$ and $\mathcal{C} := \{0,1\}^{f_2(\lambda)}$ for polynomials $f_1$ and $f_2$. We model H as a random oracle $\mathcal{H}_1 : \{0,1\}^* \to \{0,1\}^{l_1}$, available to $\mathcal{A}$.

**Theorem 10.** *For every efficient adversary $\mathcal{A}$ that makes at most $q_{LHPC}$ and $q_{RAND}$ queries to $\mathcal{O}_{LHPC}$ and $\mathcal{O}_{RAND}$, there exist adversaries $\mathcal{B}_1$, $\mathcal{B}_2$ and $\mathcal{B}_3$ such that:*

$$
\begin{aligned}
Adv_{CBC}^{IND\text{-}1\$PA\text{-}LHPC}(\mathcal{A}) \leq\ & Adv_F^{prp}(\mathcal{B}_1) \\
& + 1/2^{l_1} \\
& + \frac{q_{LHPC}}{2^{f_2(\lambda)} - 1 - q_{LHPC}} + q_{RAND} \lceil \frac{l_{max}}{f_2(\lambda)} \rceil\ 2^{-f_2(\lambda)} \\
& + Adv_F^{prp}(\mathcal{B}_2) \\
& + Adv_{CBC}^{IND\text{-}1\$PA}(\mathcal{B}_3)
\end{aligned}
$$

---

**KG**$(1^\lambda)$:
1: $K \xleftarrow{\$} \mathcal{K}$
2: **return** $K$

**Enc**$(c)$:
1: $x_1 \| ... \| x_n \leftarrow m$ s.t. $| x_i |= f_2(\lambda)\ \forall i \in \{1, ..., n\}$
2: $y_0 \leftarrow \text{SetIV}()$
3: **for all** $i \in \{1, ..., n\}$ **do**
4: $\quad \lfloor\ \ y_i \leftarrow \mathsf{F}(K, y_{i-1} \oplus x_i)$
5: $y \leftarrow y_0 \| ... \| y_i$
6: **return** $y$

**Dec**$(K, c)$:
1: $y_0 \| ... \| y_n \leftarrow c$ s.t. $| y_i |= f_2(\lambda)\ \forall i \in \{0, ..., n\}$
2: **for all** $i \in \{1, ..., n\}$ **do**
3: $\quad \lfloor\ \ x_i \leftarrow y_{i-1} \oplus \mathsf{F}^{-1}(K, y_i)$
4: $m \leftarrow x_1 \| ... \| x_n$
5: **return** $m$

---

**Fig. 25.** CBC instantiated as a symmetric encryption scheme SKE = (Kg, Enc, Dec), as defined in [5].

Let $\mathsf{Pr}_g$ denote the probability that game $g$ outputs 1. Modifications applied to the game are showcased in orange. Specific events that may cause the challenger to abort if some bad event happens are showcased in red, and are relevant only for the game where they are defined. Code that is specific to reductions is showcased in green.

**Game 0**

This is the original $\mathsf{Expt}_{CBC}^{IND\text{-}1\$PA\text{-}LHPC}$. Therefore,

$$
\mathsf{Pr}_0 = \mathsf{Adv}_{CBC}^{IND\text{-}1\$PA\text{-}LHPC}(\mathcal{A}) \tag{4}
$$

**Game 1**

Let $\mathcal{F}$ be the set of all invertible random permutations $f : \{0,1\}^{f_2(\lambda)} \to \{0,1\}^{f_2(\lambda)}$.

In this game, the challenger uniformly samples a invertible random permutation in the begginning of the experiment. Then, whenever it executes $\mathsf{Enc}(K, m)$ or $\mathsf{Dec}(K, c)$, the computation of $\mathsf{F}(K, \cdot)$ and $\mathsf{F}^{-1}(K, \cdot)$ is replaced by $f(\cdot)$ and $f^{-1}(\cdot)$, respectively.

---

$\mathsf{Game1}(\mathcal{A})$:

    $b \xleftarrow{\$} \{0, 1\}$
    $K \xleftarrow{\$} \mathsf{CBC.KG}()$
    $\mathsf{state} \leftarrow 0$
    $(m_0, m_1) \xleftarrow{\$} \mathcal{A}()$
    $c_{\mathsf{chal}} \xleftarrow{\$} \mathcal{O}_{\mathsf{LR}}(m_0, m_1)$
    $f \xleftarrow{\$} \mathcal{F}$
    $b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{LR}}, \mathcal{O}_{\mathsf{RAND}}, \mathcal{O}_{\mathsf{LHPC}}}(c_{\mathsf{chal}})$
    **return** $(b = b')$

---

**Fig. 26.** Game 1.

---

**Enc**$(c)$:

1: $x_1 \mid\mid ... \mid\mid x_n \leftarrow m$ s.t. $\mid x_i \mid = f_2(\lambda) \; \forall i \in \{1, ..., n\}$
2: $y_0 \leftarrow \mathsf{SetIV}()$
3: **for all** $i \in \{1, ..., n\}$ **do**
4:   $\lfloor \quad y_i \leftarrow f(y_{i-1} \oplus x_i)$
5: $y \leftarrow y_0 \mid\mid ... \mid\mid y_i$
6: **return** $y$

**Dec**$(K, c)$:

1: $y_0 \mid\mid ... \mid\mid y_n \leftarrow c$ s.t. $\mid y_i \mid = f_2(\lambda) \; \forall i \in \{0, ..., n\}$
2: **for all** $i \in \{1, ..., n\}$ **do**
3:   $\lfloor \quad x_i \leftarrow y_{i-1} \oplus f^{-1}(y_i)$
4: $m \leftarrow x_1 \mid\mid ... \mid\mid x_n$
5: **return** $m$

---

**Fig. 27.** Modifications to $\mathsf{CBC}$ - Game 1.

**Advantage measure** The only modification in game 1 corresponds to the replacement of the underlying invertible pseudo-random permutation in the encryption scheme with an invertible, truly random permutation. Therefore, we can bound the advantage of $\mathcal{A}$ by considering an adversary $\mathcal{B}_1$ that can distinguish between $f$ and $\mathsf{F}(K, \cdot)$ by invoking $\mathcal{A}$. Therefore:

$$\mathsf{Pr}_0 - \mathsf{Pr}_1 \leq \mathsf{Adv}_{\mathsf{F}}^{\mathsf{prp}}(\mathcal{B}_1) \tag{5}$$

**Game 2**

In this game, the challenger aborts if $\mathcal{H}_1(m_b)$ is a prefix of, but not equal to, $m_b$.

```
Game2(𝒜):
    b ←$ {0,1}
    K ←$ CBC.KG()
    state ← 0
    (m_0, m_1) ←$ 𝒜()
    bad ← false
    t ← ℋ_1(m_b)
    ▷ ⌈ |x| / f_2(λ) ⌉ is the number of blocks of size f_2(λ) for some
      string x.
    n ← ⌈ |t| / f_2(λ) ⌉, t_1 || .. || t_n ← t
    if n < ⌈ |m_b| / f_2(λ) ⌉ then
        if (t_1 || .. || t_n) = (x_1 || .. || x_n) then
            bad ← true
            abort
    c_chal ←$ 𝒪_LR(m_0, m_1)
    f ← ℱ
    b' ←$ 𝒜^{𝒪_LR, 𝒪_RAND, 𝒪_LHPC}(c_chal)
    return (b = b')
```

**Fig. 28.** Game 2.

**Advantage measure** We bound the advantage of $\mathcal{A}$ against game 2 by considering the bad event bad which happens exactly when $m_b$ is at least one block longer than $\mathcal{H}_1(m_b)$ and $\mathcal{H}_1(m_b)$ is a prefix of $m_b$.

Since $m_b$'s blocks are fixed in the beggining, for bad to occur, $\mathcal{H}_1$ must output exactly $x_1 \,||\, .. \,||\, x_n$, which happens with probability $1/2^{l_1}$.

Since games 1 and 2 proceed identically until bad, through the difference lemma, we have:

$$| \Pr_1 - \Pr_2 | \leq \Pr_2[\text{bad}] \tag{6}$$

$$\Pr_2[\text{bad}] \leq 1/2^{l_1} \tag{7}$$

Therefore,

$$| \Pr_1 - \Pr_2 | \leq 1/2^{l_1} \tag{8}$$

**Game 3**

In this game, when $b = 0$ and $\mathcal{A}$ delivers a ciphertext $c$ to $\mathcal{O}_{\text{LHPC}}$, with $c \neq c_{\text{chal}}$, and $\mathcal{H}_1(m_0) = \text{Dec}(K, c)$, the challenger returns 0.

Game3($\mathcal{A}$):
  $b \xleftarrow{\$} \{0,1\}$
  $K \xleftarrow{\$} \mathsf{CBC.KG}()$
  $\mathsf{state} \leftarrow 0$
  $(m_0, m_1) \xleftarrow{\$} \mathcal{A}()$
  $t \leftarrow \mathcal{H}_1(m_b)$
  $n \leftarrow \lceil \frac{|t|}{f_2(\lambda)} \rceil,\ t_1 \;||\; .. \;||\; t_n \leftarrow t$
  if $n < \lceil \frac{|m_b|}{f_2(\lambda)} \rceil$ then
    if $(t_1 \;||\; .. \;||\; t_n) = (x_1 \;||\; .. \;||\; x_n)$
    then
      abort
  $c_{\mathsf{chal}} \xleftarrow{\$} \mathcal{O}_{\mathsf{LR}}(m_0, m_1)$
  $f \leftarrow \mathcal{F}$
  $\mathsf{bad} \leftarrow \mathsf{false}$
  $b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{LR}}, \mathcal{O}_{\mathsf{RAND}}, \mathcal{O}_{\mathsf{LHPC}}}(c_{\mathsf{chal}})$
  return $(b = b')$

$\mathcal{O}_{\mathsf{LHPC}}(c)$:
  if $c = c_{\mathsf{chal}}$ then
    return $\perp$
  if $b = 0$ then
    if $\mathcal{H}_1(m_0) = \mathsf{CBC.Dec}(K, c)$ then
      $\mathsf{bad} \leftarrow \mathsf{true}$
      return $0$
  return $\mathcal{H}_1(m_0) = \mathsf{CBC.Dec}(K, c)$

**Fig. 29.** Game 3.

**Advantage measure** We bound the advantage of $\mathcal{A}$ against game 3 by considering the bad event $\mathsf{bad}$ which happens exactly when $b = 0$ and $\mathcal{A}$ delivers a ciphertext $c$ to $\mathcal{O}_{\mathsf{LHPC}}$ which correctly decrypts into $\mathcal{H}_1(m_0)$ and is different from the challenge ciphertext $c_{\mathsf{chal}}$.

We know from the definition of $\mathcal{H}_1$ that the target ciphertext $c$ must have length *exactly* $\lceil \frac{l_1}{f_2(\lambda)} \rceil + 1$. Let $n = \lceil \frac{l_1}{f_2(\lambda)} \rceil + 1$.

In order for $\mathcal{A}$ to produce $c = y_0 \;||\; ... \;||\; y_n$, for a message $m = \mathcal{H}_1(m_0) = x_1 \;||\; ... \;||\; x_n$, it must be able to produce, depending on $m_0$, one or more of its blocks without having access to $f$ or $f^{-1}$.

If $m_0 \neq \mathcal{H}_1(m_0)$, and $m_0$ and $\mathcal{H}_1(m_0)$ share the first $p$ blocks, for $0 \leq p < n$, then $\mathcal{A}$ must be able to generate all $n - p$ blocks of the target ciphertext $c$. The best possible case for $\mathcal{A}$ is when $p = n - 1$, i.e., when $\mathcal{H}_1(m_0)$ shares all of its blocks with $m_0$ except for the last one. Therefore, $\mathcal{A}$ only has to produce the last block $y_n$ for $c$ to decrypt into $m_0$.

However, since $y_n = f(y_{n-1} \oplus x_n)$, and $\mathcal{A}$ does not have access to $f$ or $f^{-1}$, $\mathcal{A}$ has no information about $y_n$, and therefore has only two options: either guess $y_n$ directly, or obtain it through the $\mathcal{O}_{\mathsf{RAND}}$ oracle.

The probability that $\mathcal{A}$ guesses $y_n$ directly can be upper bounded by considering $q_{\mathsf{LHPC}}$ as the maximum number of queries to $\mathcal{O}_{\mathsf{LHPC}}$, and that for the first $\mathcal{O}_{\mathsf{LHPC}}$ query, $\mathcal{A}$ can guess $y_n$ with probability at most $\frac{1}{2^{f_2(\lambda)} - 1}$ (in case $m_0$ has more blocks than $\mathcal{H}_1(m_0)$, in which case the last block of $c$ must be different from the first distinct block of $c_{\mathsf{chal}}$), on the second attempt with probability $\frac{1}{2^{f_2(\lambda)} - 2}$ and so on. Each query attempted by $\mathcal{A}$ effectively reduces the probability space to guess the correct block $y_n$. Therefore, $\mathcal{A}$ can guess $y_1$ with probability at most

$$\sum_{i=0}^{q_{\mathsf{LHPC}}} \frac{1}{2^{f_2(\lambda)} - 1 - i} \leq \frac{q_{\mathsf{LHPC}}}{2^{f_2(\lambda)} - 1 - q_{\mathsf{LHPC}}} \tag{9}$$

The probability that $\mathcal{O}_{\mathsf{RAND}}$ inputs $y_{n-1} \oplus x_n$ to $f$ is, at most, $2^{-f_2(\lambda)}$ per block per query, since each query to $\mathcal{O}_{\mathsf{RAND}}$ can produce the same input to $f$ more than once. Therefore, we can upper bound the probability of $f(y_{n-1} \oplus x_n)$ occuring throughout the game as

$$q_{\mathsf{RAND}} \left\lceil \frac{l_{\max}}{f_2(\lambda)} \right\rceil 2^{-f_2(\lambda)} \tag{10}$$

, where $l_{\max}$ is the largest length $l$ given to $\mathcal{O}_{\mathsf{RAND}}$ by $\mathcal{A}$, and $\left\lceil \frac{l_{\max}}{f_2(\lambda)} \right\rceil$ is the maximum number of times each query to $\mathcal{O}_{\mathsf{RAND}}$ calls $f$.

In these circunstances, $\mathcal{A}$ can produce $c$ such that $\mathcal{H}_1(m_0) = \mathsf{Dec}(K, c)$ with a maximum probability of

$$\frac{q_{\mathsf{LHPC}}}{2^{f_2(\lambda)} - 1 - q_{\mathsf{LHPC}}} + q_{\mathsf{RAND}} \left\lceil \frac{l_{\max}}{f_2(\lambda)} \right\rceil 2^{-f_2(\lambda)} \tag{11}$$

If $m_0 = \mathcal{H}_1(m_0)$, then $\mathcal{A}$ must produce a target ciphertext $c$ that is different from $c_{\mathsf{chal}}$. However, in order to produce this $c$, $\mathcal{A}$ must use a new IV.

Recall that $\mathcal{A}$ must produce $c = y_0 \mathbin{||} ... \mathbin{||} y_n$ such that $\mathsf{Dec}(K, c) = \mathsf{Dec}(K, c_{\mathsf{chal}})$ but $c$ cannot share all of its blocks with $c_{\mathsf{chal}}$, and that in CBC mode, each ciphertext block $y_i$ is calculated as $y_i \leftarrow f(y_{i-1} \oplus x_i), \forall i \in \{1, ..., n\}$. If $c$ and $c_{\mathsf{chal}}$ share the same first block $y_0$ (the IV), then $y_1 = f(y_0 \oplus x_1)$ will always be the same when calculated with the same random permutation $f$ over the same message $m = x_1 \mathbin{||} ... \mathbin{||} x_n$. This follows for all $y_i$, which always depends on the previous block $y_{i-1}$. Therefore, by sharing the first block, $c$ necessarily shares all of its blocks with $c_{\mathsf{chal}}$, and thus is either the same as or a prefix of $c_{\mathsf{chal}}$.

After generating a new IV, which is block $y_0$, $\mathcal{A}$ must be able to create all remaining $\left\lceil \frac{l_1}{f_2(\lambda)} \right\rceil$ blocks, which are calculated by $f(y_{i-1} \oplus x_i), \forall i \in 1, .., n$, without having access to $f$ or $f^{-1}$. The best possible case for $\mathcal{A}$ is when $\left\lceil \frac{l_1}{f_2(\lambda)} \right\rceil = 1$, and thus $n = 2$, which means $\mathcal{A}$ must only produce $y_1$, which is also the last block. From the previous case, we know that $\mathcal{A}$ can produce $y_1$ with probability at most

$$\frac{q_{\mathsf{LHPC}}}{2^{f_2(\lambda)} - 1 - q_{\mathsf{LHPC}}} + q_{\mathsf{RAND}} \left\lceil \frac{l_{\max}}{f_2(\lambda)} \right\rceil 2^{-f_2(\lambda)} \tag{12}$$

Notice that each of these cases is mutually exclusive, but that the absolute best case scenario for $\mathcal{A}$ in either case can be upper bounded by equations 11/12.

Since games 2 and 3 proceed identically until **bad** happens, through the difference lemma, we have:

$$\mid \mathsf{Pr}_2 - \mathsf{Pr}_3 \mid \leq \mathsf{Pr}_3[\mathsf{bad}] \tag{13}$$

Since **bad** can only happen if $b = 0 \wedge \mathcal{H}_1(m_0) = \mathsf{Dec}(K, c)$, we have:

$$\mathsf{Pr}_3[\mathsf{bad}] \leq 2^{-1} \left( \frac{q_{\mathsf{LHPC}}}{2^{f_2(\lambda)} - 1 - q_{\mathsf{LHPC}}} + q_{\mathsf{RAND}} \left\lceil \frac{l_{\max}}{f_2(\lambda)} \right\rceil 2^{-f_2(\lambda)} \right) \tag{14}$$

Therefore,

$$\mid \mathsf{Pr}_2 - \mathsf{Pr}_3 \mid \leq 2^{-1} \frac{q_{\mathsf{LHPC}}}{2^{f_2(\lambda)} - 1 - q_{\mathsf{LHPC}}} + 2^{-1} q_{\mathsf{RAND}} \left\lceil \frac{l_{\max}}{f_2(\lambda)} \right\rceil 2^{-f_2(\lambda)} \tag{15}$$

## Game 4

In this game, when $b = 1$ and $\mathcal{A}$ delivers a ciphertext $c$ to $\mathcal{O}_{\text{LHPC}}$, with $c \neq c_{\text{chal}}$, and $\mathcal{H}_1(m_0) = \text{Dec}(K, c)$, the challenger returns 0.



Game4($\mathcal{A}$):
  $b \xleftarrow{\$} \{0, 1\}$
  $K \xleftarrow{\$} \text{CBC.KG}()$
  state $\leftarrow 0$
  $(m_0, m_1) \xleftarrow{\$} \mathcal{A}()$
  $t \leftarrow \mathcal{H}_1(m_b)$
  $n \leftarrow \lceil \frac{|t|}{f_2(\lambda)} \rceil$, $t_1 \| .. \| t_n \leftarrow t$
  if $n < \lceil \frac{|m_b|}{f_2(\lambda)} \rceil$ then
    if $(t_1 \| .. \| t_n) = (x_1 \| .. \| x_n)$
    then
      abort
  $c_{\text{chal}} \xleftarrow{\$} \mathcal{O}_{\text{LR}}(m_0, m_1)$
  $f \leftarrow \mathcal{F}$
  bad $\leftarrow$ false
  $b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\text{LR}}, \mathcal{O}_{\text{RAND}}, \mathcal{O}_{\text{LHPC}}}(c_{\text{chal}})$
  return $(b = b')$

$\mathcal{O}_{\text{LHPC}}(c)$:
  if $c = c_{\text{chal}}$ then
    return $\perp$
  if $b = 0$ then
    if $\mathcal{H}_1(m_0) = \text{CBC.Dec}(K, c)$ then
      return 0
  if $b = 1$ then
    if $\mathcal{H}_1(m_0) = \text{CBC.Dec}(K, c)$ then
      bad $\leftarrow$ true
      return 0
  return $\mathcal{H}_1(m_0) = \text{CBC.Dec}(K, c)$

**Fig. 30.** Game 4.

**Advantage measure** We bound the advantage of $\mathcal{A}$ against game 4 by considering the bad event bad which happens exactly when $b = 1$ and $\mathcal{A}$ delivers a ciphertext $c$ to $\mathcal{O}_{\text{LHPC}}$ which correctly decrypts into $\mathcal{H}_1(m_0)$.

The analysis is identical to game 3.

Let $n = \lceil \frac{l_1}{f_2(\lambda)} \rceil + 1$. Unless $\mathcal{A}$ provides a message $m_1$ such that $m_1 = \mathcal{H}_1(m_0)$, which results in $\mathcal{A}$ having to generate all $n$ blocks of ciphertext $c$, $\mathcal{A}$ must generate, at least, one block $y_n$ of ciphertext $c$. This happens with probability at most

$$\frac{q_{\text{LHPC}}}{2^{f_2(\lambda)} - 1 - q_{\text{LHPC}}} + q_{\text{RAND}} \lceil \frac{l_{\text{max}}}{f_2(\lambda)} \rceil 2^{-f_2(\lambda)} \tag{16}$$

Since games 3 and 4 proceed identically until bad happens, through the difference lemma, we have:

$$|\Pr_3 - \Pr_4| \leq \Pr_4[\text{bad}] \tag{17}$$

Since bad can only happen if $b = 1 \wedge \mathcal{H}_1(m_0) = \text{Dec}(K, c)$, we have:

$$\Pr_3[\text{bad}] \leq 2^{-1} \left( \frac{q_{\text{LHPC}}}{2^{f_2(\lambda)} - 1 - q_{\text{LHPC}}} + q_{\text{RAND}} \lceil \frac{l_{\text{max}}}{f_2(\lambda)} \rceil 2^{-f_2(\lambda)} \right) \tag{18}$$

Therefore,

$$|\Pr_3 - \Pr_4| \leq 2^{-1} \frac{q_{\text{LHPC}}}{2^{f_2(\lambda)} - 1 - q_{\text{LHPC}}} + 2^{-1} q_{\text{RAND}} \lceil \frac{l_{\text{max}}}{f_2(\lambda)} \rceil 2^{-f_2(\lambda)} \tag{19}$$

## Game 5

In this game, whenever $\mathcal{A}$ calls oracle $\mathcal{O}_{\mathsf{LHPC}}$ with a ciphertext $c \neq c_{\mathsf{chal}}$, the oracle returns 0.

$\mathcal{O}_{\mathsf{LHPC}}(c)$:
    **if** $c = c_{\mathsf{chal}}$ **then**
        $\lfloor$ **return** $\bot$
    **return** 0

**Fig. 31.** Game 5.

**Advantage measure** In the previous game, the oracle $\mathcal{O}_{\mathsf{LHPC}}$ always returns 0 to $\mathcal{A}$ when $c \neq c_{\mathsf{chal}}$, regardless of $b$. Therefore, this game is functionally identical to game 4, and we have:

$$\mathsf{Pr}_4 = \mathsf{Pr}_5 \tag{20}$$

## Game 6

In this game, whenever the challenger executes $\mathsf{Enc}(K, m)$ or $\mathsf{Dec}(K, c)$, the computation of $f(\cdot)$ and $f^{-1}(\cdot)$ is replaced by $\mathsf{F}(K, \cdot)$ and $\mathsf{F}^{-1}(K, \cdot)$, respectively.

**Enc**$(c)$:
1: $x_1 \;||\; ... \;||\; x_n \leftarrow m$ s.t. $|\; x_i \;| = f_2(\lambda) \; \forall i \in \{1, ..., n\}$
2: $y_0 \leftarrow \mathsf{SetIV}()$
3: **for all** $i \in \{1, ..., n\}$ **do**
4: $\lfloor \quad y_i \leftarrow \mathsf{F}(K, y_{i-1} \oplus x_i)$
5: $y \leftarrow y_0 \;||\; ... \;||\; y_i$
6: **return** $y$

**Dec**$(K, c)$:
1: $y_0 \;||\; ... \;||\; y_n \leftarrow c$ s.t. $|\; y_i \;| = f_2(\lambda) \; \forall i \in \{0, ..., n\}$
2: **for all** $i \in \{1, ..., n\}$ **do**
3: $\lfloor \quad x_i \leftarrow y_{i-1} \oplus \mathsf{F}^{-1}(K, y_i)$
4: $m \leftarrow x_1 \;||\; ... \;||\; x_n$
5: **return** $m$

**Fig. 32.** Modifications to $\mathsf{CBC}$ - Game 6.

**Advantage measure** The only modification in game 6 corresponds to the replacement of the underlying invertible truly random permutation in the encryption scheme with an invertible, pseudo-random permutation. Therefore, we can bound the advantage of $\mathcal{A}$ by considering an adversary $\mathcal{B}_2$ that can distinguish between $\mathsf{F}(K, \cdot)$ and $f$ by invoking $\mathcal{A}$. Therefore:

$$\mathsf{Pr}_5 - \mathsf{Pr}_6 \leq \mathsf{Adv}_{\mathsf{F}}^{\mathsf{prp}}(\mathcal{B}_2) \tag{21}$$

## Final Analysis

We can now calculate an upper bound for the advantage of $\mathcal{A}$ against game 6 by reduction to the $\mathsf{IND\text{-}1\$PA}$ security of $\mathsf{CBC}$.

Let $\mathcal{B}_3$ be an adversary against the IND-1\$PA security of CBC. We show that if $\mathcal{A}$ wins game 6, then $\mathcal{B}_3$ wins the IND-1\$PA game.



$\mathcal{B}_3(\mathcal{A}, \mathcal{O}_{\mathsf{LR}}^{\mathsf{IND\text{-}1\$PA}}, \mathcal{O}_{\mathsf{RAND}}^{\mathsf{IND\text{-}1\$PA}})$:

  $(m_0, m_1) \xleftarrow{\$} \mathcal{A}()$
  $t \leftarrow \mathcal{H}_1(m_b)$
  $n \leftarrow \lceil \frac{|t|}{f_2(\lambda)} \rceil, \ t_1 \ || \ .. \ || \ t_n \leftarrow t$
  **if** $n < \lceil \frac{|m_b|}{f_2(\lambda)} \rceil$ **then**
   **if** $(t_1 \ || \ .. \ || \ t_n) = (x_1 \ || \ .. \ || \ x_n)$
   **then**
    abort
  $c_{\mathsf{chal}} \xleftarrow{\$} \mathcal{O}_{\mathsf{LR}}(m_0, m_1)$
  $b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{LR}}, \mathcal{O}_{\mathsf{RAND}}, \mathcal{O}_{\mathsf{LHPC}}}(c_{\mathsf{chal}})$
  **return** $(b')$

$\mathcal{O}_{\mathsf{LR}}(m_0, m_1)$:

  $c_{\mathsf{chal}} \leftarrow \mathcal{O}_{\mathsf{LR}}^{\mathsf{IND\text{-}1\$PA}}(m_0, m_1)$
  **return** $c_{\mathsf{chal}}$

$\mathcal{O}_{\mathsf{LHPC}}(c)$:

  **if** $c = c_{\mathsf{chal}}$ **then**
   **return** $\perp$
  **return** $0$

$\mathcal{O}_{\mathsf{RAND}}(l)$:

  $(m_0', m_1', c') \leftarrow \mathcal{O}_{\mathsf{RAND}}^{\mathsf{IND\text{-}1\$PA}}(l)$
  **return** $(m_0', m_1', c')$

**Fig. 33.** Game 6 simulated by $\mathcal{B}_3$, with corresponding Left-Right oracle $\mathcal{O}_{\mathsf{LR}}$, RAND oracle $\mathcal{O}_{\mathsf{RAND}}$ and LHPC oracle $\mathcal{O}_{\mathsf{LHPC}}$.

**Advantage measure** From the definition of $\mathcal{B}_3$, if $\mathcal{A}$ wins game 6, then $\mathcal{B}_3$ wins against the IND-1\$PA security of CBC.

$$\mathsf{Pr}_6 \leq \mathsf{Pr}_{\mathsf{CBC}}^{\mathsf{IND\text{-}1\$PA}}(\mathcal{B}_3) \tag{22}$$

Note that the running times of all adversaries are close to that of $\mathcal{A}$.
This concludes the proof.

# L   CTAP 2.1+ Authentication Proof

In this section, we give a fully detailed, code-based proof for the authentication security of CTAP 2.1+, as an instance of mPACA, with a sequence of games. We base our modifications for each game on the mPACA model oracles and CTAP 2.1+ syntax as presented in Sections 4 and B, respectively.

Let $\mathsf{Pr}_g$ denote the probability that game $g$ outputs 1. Each game's modifications that are *persistent* throughout the proof are showcased in orange. Modifications that exist only when employing a hybrid argument or a reduction, and therefore are only relevant to a specific game, are shown in green. Finally, modifications in red flag **bad** events or the end of a reduction and, as a consequence, are also relevant only to their respective game.

## Game 0

This game is the original $\mathsf{Expt}_{\mathsf{CTAP\ 2.1+}}^{\mathsf{SUF\text{-}t}}(\mathcal{A})$ security experiment. Therefore,

$$\mathsf{Pr}_0 = \mathsf{Adv}_{\mathsf{CTAP\ 2.1+}}^{\mathsf{SUF\text{-}t}}(\mathcal{A}) \tag{23}$$

## Game 1

In this game we replace every key $K_1$ and $K_2$ derived from $Z$ by all passive ECDH key exchange sessions with two independent and uniformly random session keys $\tilde{K}_1$ and $\tilde{K}_2$, and bound the advantage of $\mathcal{A}$ by reducing to the sCDH security of ECDH.

We maintain two structures $\mathcal{L}_{\mathsf{sCDH}}$ and $\mathcal{L}_{\mathcal{H}_2}$ respectively to save ocurrences of client and token public keys in order to maintain consistency throughout the game and to save adversary queries to random oracle $\mathcal{H}_2$. Every execution of encapsulate, which happens only on clients, will first check if the current pair of public keys has appeared before. If it hasn't, the experiment samples the two session keys $\tilde{K}_1$ and $\tilde{K}_2$ and saves them in $\mathcal{L}_{\mathsf{sCDH}}$. Otherwise, the challenger uses the $\tilde{K}_1$ and $\tilde{K}_2$ which were previously stored in $\mathcal{L}_{\mathsf{sCDH}}$. On the token side, everytime decapsulate executes, it performs the same check. If the pair does not exist, then the public key received must not have been generated by an honest client, since $\mathcal{L}_{\mathsf{sCDH}}$ is only ever set by the client in encapsulate. In that situation, the regular ECDH key derivation is used. If it does exist, then both keys have already appeared in the experiment before, in a previous execution of encapsulate, which means the same $\tilde{K}_1$ and $\tilde{K}_2$ must be used, in order to maintain consistency. It is clear that, unless $\mathcal{A}$ ever queried $\mathcal{H}_2$ with the shared secret $Z$ derived in a Setup or Execute query, Game 1 proceeds identically to Game 0.



**Fig. 34.** Game 1 on the left, and $\mathcal{H}_2$ wrapper $\mathsf{Adv}\mathcal{H}_2$ on the right - Game 1. $\mathsf{Adv}\mathcal{H}_2$ is a wrapper that $\mathcal{A}$ can use to call the random oracle, so that we can distinguish calls to $\mathcal{H}_2$ made specifically by $\mathcal{A}$.



**Fig. 35.** Modifications to PIN/UV Auth Protocol functions encapsulate and decapsulate - Game 1. Variables $sk$ and $pk$ refer to the stored keys being used by the client session or the token session.

**Hybrid Argument** We bound the advantage of $\mathcal{A}$ against Game 1 by creating $n_{\mathsf{TokenPK}}$ hybrid games between Game 0 and Game 1, where $n_{\mathsf{TokenPK}}$ is the total number of token public keys sampled throughout the experiment that are output to $\mathcal{A}$ in Setup, Execute and Send-Bind-T queries, via the obtainSharedSecret-T function. We then define a bad event bad on game $0.i$, where the game aborts and $\mathcal{A}$ loses if it queries $\mathcal{H}_2$ with $u_1 = Z =$

XCoordinateOf($sk \cdot pk_{\text{token}}^i$), and $pk_{\text{token}}^i$ and $c$ exist in $\mathcal{L}_{\text{sCDH}}$ at any point in the game, with $pk_{\text{token}}^i$ as the $i^{th}$ token public key and $sk$ and $c$ any client private and public key generated by a client session that received $pk_{\text{token}}^i$.

We use the notation from regular Diffie-Hellman instead of Elliptic Curve Diffie-Hellman to simplify the code and make it more legible. For example, instead of $Z \leftarrow$ XCoordinateOf($sk \cdot pk'$), we use $Z \leftarrow (pk')^{sk}$ during this hybrid and the reduction to sCDH. However, every operation used in this proof uses ECDH.

---

**Game0.$i(\mathcal{A})$:**
$\mathcal{L}_{\text{authC}}, \mathcal{L}_{\text{authT}} \leftarrow \emptyset, \mathcal{L}_{\text{valid}} \leftarrow \emptyset, \mathcal{L}_{\text{corrupt}} \leftarrow \emptyset$
$\mathcal{L}_{\text{sCDH}}, \mathcal{L}_{\mathcal{H}_2} \leftarrow \emptyset$
$\text{hy} \leftarrow i, \mathcal{L}_{hy} \leftarrow \emptyset$
$(pk_{\text{token}}^i, sk_{\text{token}}^i) \leftarrow (\bot, \bot)$
$\text{bad} \leftarrow \text{false}$
$\text{win-SUF-t} \leftarrow 0$
$() \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(1^\lambda)$
**return** win-SUF-t

**Adv$\mathcal{H}_2$ $(u_1, u_2)$:**
$v \leftarrow \mathcal{H}_2(u_1, u_2)$
**for all** $(pk_T, pk_C, \_) \in \mathcal{L}_{\text{sCDH}}$ s.t.
$pk_T = pk_{\text{token}}^i$ **do**
  **if** $u_1 = (pk_C)^{sk_{\text{token}}^i}$ **then**
    $\text{bad} \leftarrow \text{true}$
    abort
$\mathcal{L}_{\mathcal{H}_2}[u_1, u_2] \leftarrow v$
**return** $v$

**Fig. 36.** Hybrid game $0.i$ on the left and random oracle wrapper Adv$\mathcal{H}_2$ on the right.

---

**obtainSharedSecret-T($\pi_T^i$, puvProtocol):**
**if** puvProtocol $\notin \text{st}_T$.puvProtocolList **then**
  **return** $\bot$
$pk_T \leftarrow \text{st}_T$.puvProtocol.getPublicKey()
**if** $(T, pk_T) \notin \mathcal{L}_{\text{hy}}$ **then**
  $\mathcal{L}_{\text{hy}} \leftarrow \mathcal{L}_{\text{hy}} \cup (T, pk_T)$
  **if** $\text{size}(\mathcal{L}_{\text{hy}}) = \text{hy}$ **then**
    $(pk_{\text{token}}^i, sk_{\text{token}}^i) \leftarrow$
    $(pk_T, \text{st}_T.\text{puvProtocol}.sk)$
selectpuvProtocol $\leftarrow$ puvProtocolList
$\pi_T^i.\text{st}_{\text{exe}} \leftarrow$ waiting
$\pi_T^i.\text{sid} \leftarrow \pi_T^i.\text{sid} \parallel$ puvProtocol $\parallel pk_T$
**return** $pk_T$

**encapsulate $(pk')$:**
**if** $(\_, pk') \in \mathcal{L}_{hy}[0 : \text{hy}[$ **then**
  **if** $\mathcal{L}_{\text{sCDH}}[pk', pk] = \bot$ **then**
    $\tilde{K}_1 \xleftarrow{\$} \{0,1\}^{2\lambda}, \tilde{K}_2 \xleftarrow{\$} \{0,1\}^{2\lambda}$
    $\tilde{K} \leftarrow (\tilde{K}_1, \tilde{K}_2)$
    $\mathcal{L}_{\text{sCDH}}[pk', pk] \leftarrow \tilde{K}$
    **if** $pk' = pk_{\text{token}}^i$ **then**
      **if** $\mathcal{L}_{\mathcal{H}_2}[(pk')^{sk}, \_] \neq \bot$ **then**
        $\text{bad} \leftarrow \text{true}$
        abort
  **else**
    $\tilde{K} \leftarrow \mathcal{L}_{\text{sCDH}}[pk', pk]$
  $K \leftarrow \tilde{K}$
  $c \leftarrow pk$
  **return** $(c, K)$
**if** $(\_, pk') \in \mathcal{L}_{hy}[\text{hy} : n_{\text{TokenPK}}[$ **then**
  $Z \leftarrow (pk')^{sk}$
  $K_1 \leftarrow \mathcal{H}_2(Z, \text{"CTAP2 HMAC KEY"})$
  $K_2 \leftarrow \mathcal{H}_2(Z, \text{"CTAP2 AES KEY"})$
  $K \leftarrow (K_1, K_2)$
  $c \leftarrow pk$
  **return** $(c, K)$

**decapsulate $(c)$:**
**if** $(\_, pk) \in \mathcal{L}_{hy}[0 : \text{hy}[$ **then**
  $\tilde{K} \leftarrow \mathcal{L}_{\text{sCDH}}[pk, c]$
  **if** $\tilde{K} \neq \bot$ **then**
    $K \leftarrow \tilde{K}$
    **return** $K$
  $Z \leftarrow c^{sk}$
  $K_1 \leftarrow \mathcal{H}_2(Z, \text{"CTAP2 HMAC KEY"})$
  $K_2 \leftarrow \mathcal{H}_2(Z, \text{"CTAP2 AES KEY"})$
  $K \leftarrow (K_1, K_2)$
  **return** $K$
**if** $(\_, pk) \in \mathcal{L}_{hy}[\text{hy} : n_{\text{TokenPK}}[$ **then**
  $Z \leftarrow c^{sk}$
  $K_1 \leftarrow \mathcal{H}_2(Z, \text{"CTAP2 HMAC KEY"})$
  $K_2 \leftarrow \mathcal{H}_2(Z, \text{"CTAP2 AES KEY"})$
  $K \leftarrow (K_1, K_2)$
  **return** $K$

**Fig. 37.** obtainSharedSecret-T on the left and PIN/UV Auth Protocol encapsulate and decapsulate functions on the right, from hybrid game $0.i$.

---

**Reduction to the sCDH security of ECDH** We now construct an adversary $\mathcal{B}_1^i$ that simulates game $0.i$ to $\mathcal{A}$, in such a way that when bad happens, $\mathcal{B}_1^i$ wins against the sCDH security of ECDH.

Throughout the simulation of game $0.i$, $\mathcal{B}_1^i$ sets the $i^{th}$ token public key as the sCDH challenge public key $g^a$ and any client public key generated by a client session that receives

$g^a$ to be $g^{b+r_w}$, for $r_w \leftarrow \mathbb{Z}_q$ and $w \geq 1$. In such client sessions, $\mathcal{B}_1^i$ checks if $u_1 = Z$ is in $\mathcal{L}_{\mathsf{sCDH}}$ using its $\mathcal{O}_a$ oracle. Additionally, everytime $\mathcal{A}$ calls random oracle $\mathcal{H}_2$ (always through $\mathsf{Adv}\mathcal{H}_2$) $\mathcal{B}_1^i$ also checks if for any public key pair in $\mathcal{L}_{\mathsf{sCDH}}$ that includes $g^a$, $u_1 = Z$.

It is not hard to see that, whenever bad happens during $\mathcal{B}_1^i$'s simulation, $\mathcal{B}_1^i$ wins by using $\mathcal{A}$'s input to the random oracle $\mathcal{H}_2$. Indeed, if $\mathcal{A}$ ever calls $\mathcal{H}_2$ with input $u_1 = Z = (g^{b+r_w})^a, \forall r_w \in \mathcal{L}_{r_w}$, then $\mathcal{B}_1^i$ can output $Z' \leftarrow u_1 \cdot (g^a)^{-r_w} \equiv (g^{b+r_w})^a \cdot (g^a)^{-r_w} \equiv g^{ab}$ and win the $\mathsf{sCDH}$ game.

---

**Challenger$_{\mathsf{ECDH}}^{\mathsf{sCDH}}$:**
   $(a, g^a) \xleftarrow{\$} \mathsf{ECDH.KG}()$
   $(b, g^b) \xleftarrow{\$} \mathsf{ECDH.KG}()$
   $Z' \leftarrow \mathcal{A}^{\mathcal{O}_a}(g, g^a, g^b)$
   $Z \leftarrow (g^b)^a$
   $\mathsf{win\text{-}sCDH} \leftarrow (Z = Z')$
   **return** $\mathsf{win\text{-}sCDH}$

**$\mathcal{O}_a(c, Z)$:**
   $Z' \leftarrow c^a$
   **return** $(Z = Z')$

**$\mathcal{B}_1^i(g, g^a, g^b, \mathcal{A}, \mathcal{O}_a)$:**
   $\mathcal{L}_{\mathsf{authC}}, \mathcal{L}_{\mathsf{authT}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{valid}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{corrupt}} \leftarrow \emptyset$
   $\mathcal{L}_{\mathsf{sCDH}}, \mathcal{L}_{\mathcal{H}_2} \leftarrow \emptyset$
   $\mathsf{hy} \leftarrow i, \mathcal{L}_{hy} \leftarrow \emptyset$
   $(pk_{\mathsf{token}}^i, sk_{\mathsf{token}}^i) \leftarrow (\bot, \bot)$
   $\mathcal{L}_{r_w}, \mathcal{L}_{\mathsf{Fake}\mathcal{H}_2} \leftarrow \emptyset$
   $Z' \xleftarrow{\$} \mathbb{Z}_q$
   $\mathsf{win\text{-}SUF\text{-}t} \leftarrow 0$
   $() \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(1^\lambda)$
   ▷ *If bad ever happened, $Z'$ has the correct value*
   *because it was set during the game simulation*
   *from $\mathcal{B}_1^i$.*
   **return** $Z'$

**Fig. 38.** $\mathsf{sCDH}$ Challenger and oracle on the left and adversary $\mathcal{B}_1^i$'s simulation of game $0.i$ on the right.

---

**$\mathcal{H}_2(u_1, u_2)$:**
   **if** $t_1[u_1, u_2] \neq \bot$ **then**
     **return** $t_1[u_1, u_2]$
   **else**
     **for** $(c, (\tilde{K}_1, \tilde{K}_2)) \in \mathcal{L}_{\mathsf{Fake}\mathcal{H}_2}$ **do**
       **if** $\mathcal{O}_a(c, u_1) = \mathsf{true}$ **then**
         **if** $u_2 = $ "CTAP2 HMAC KEY" **then**
           $t_1[u_1, u_2] \leftarrow \tilde{K}_1$
           **return** $\tilde{K}_1$
         **else if** $u_2 = $ "CTAP2 AES KEY"
         **then**
           $t_1[u_1, u_2] \leftarrow \tilde{K}_2$
           **return** $\tilde{K}_2$
     $v \xleftarrow{\$} \{0, 1\}^{2\lambda}$
     $t_1[u_1, u_2] \leftarrow v$
     **return** $v$

**$\mathsf{Adv}\mathcal{H}_2(u_1, u_2)$:**
   $v \leftarrow \mathcal{H}_2(u_1, u_2)$
   **for all** $(r_w, g^b \cdot g^{r_w}) \in \mathcal{L}_{r_w}$ **do**
     **if** $\mathcal{L}_{\mathsf{sCDH}}[g^a, g^b \cdot g^{r_w}] \neq \bot$ **then**
       **if** $\mathcal{O}_a(g^b \cdot g^{r_w}, u_1) = \mathsf{true}$ **then**
         $Z' = u_1 \cdot (g^a)^{-r_w}$
         $\mathsf{win\text{-}sCDH}$
   $\mathcal{L}_{\mathcal{H}_2}[u_1, u_2] \leftarrow v$
   **return** $v$

**Fig. 39.** Random Oracle $\mathcal{H}_2$ and wrapper $\mathsf{Adv}\mathcal{H}_2$ simulated by $\mathcal{B}_1^i$.

---

**Advantage measure** By the definition of $\mathcal{B}_1^i$, if bad happens, then $\mathcal{A}$ found the solution to the problem given by the $\mathsf{sCDH}$ challenger to $\mathcal{B}_1^i$, i.e., $g^{ab}$. Therefore:

$$\mathsf{Pr}_{0.i}[\mathsf{bad}] \leq \mathsf{Adv}_{\mathsf{ECDH}}^{\mathsf{sCDH}}(\mathcal{B}_1^i) \qquad (24)$$

Since games $0.i$ and $0.(i-1)$ are identical until bad happens, through the difference lemma, we have:

```
obtainSharedSecret-T(π_T^i, puvProtocol):
    if puvProtocol ∉ st_T.puvProtocolList then
        └ return ⊥
    pk_T ← st_T.puvProtocol.getPublicKey()
    if (T, pk_T) ∉ L_hy then
        │ L_hy ← L_hy ∪ (T, pk_T)
        │ if size(L_hy) = hy then
        └ └ (pk_token^i, sk_token^i) ← (g^a, _)
    selectpuvProtocol ← puvProtocolList
    π_T^i.st_exe ← waiting
    π_T^i.sid ← π_T^i.sid || puvProtocol || pk_T
    return pk_T
```

```
encapsulate (pk'):
    if (_, pk') ∈ L_hy[0 : hy[ then
        │ if pk' = pk_token^i then
        │     │ w ← size(L_r_w) + 1
        │     │ r_w ←$ Z_q
        │     │ pk ← g^b · g^r_w
        │     └ L_r_w ← L_r_w ∪ {(r_w, pk)}
        │ if L_sCDH[pk', pk] = ⊥ then
        │     │ K̃_1 ←$ {0,1}^2λ, K̃_2 ←$ {0,1}^2λ
        │     │ K̃ ← (K̃_1, K̃_2)
        │     │ L_sCDH[pk', pk] ← K̃
        │     │ if pk' = pk_token^i then
        │     │     │ for all (u_1, u_2, v) ∈ L_H_2 do
        │     │     │     │ if O_a(pk, u_1) = true then
        │     │     │     │     ▷ u_1 = (g^b · g^r_w)^a
        │     │     │     │     Z' = u_1 · (g^a)^{-r_w}
        │     │     │     └     win-sCDH
        │     else
        │     └ K̃ ← L_sCDH[pk', pk]
        │ K ← K̃
        │ c ← pk
        └ return (c, K)
    if (_, pk') ∈ L_hy[hy : n_TokenPK[ then
        │ Z ← (pk')^sk
        │ K_1 ← H_2(Z, "CTAP2 HMAC KEY")
        │ K_2 ← H_2(Z, "CTAP2 AES KEY")
        │ K ← (K_1, K_2)
        │ c ← pk
        └ return (c, K)
```

```
decapsulate (c):
    if (_, pk) ∈ L_hy[0 : hy[ then
        │ K̃ ← L_sCDH[pk, c]
        │ if K̃ ≠ ⊥ then
        │     │ K ← K̃
        │     └ return K
        │ if pk = pk_token^i then
        │     │ for all (u_1, u_2, v) ∈ L_H_2 do
        │     │     │ if O_a(c, u_1) = true then
        │     │     │     │ K_1 ← H_2(u_1, "CTAP2
        │     │     │     │     HMAC KEY")
        │     │     │     │ K_2 ← H_2(u_1, "CTAP2 AES
        │     │     │     │     KEY")
        │     │     │     │ K ← (K_1, K_2)
        │     │     │     └ return K
        │     │ K̃_1 ←$ {0,1}^l3, K̃_2 ←$ {0,1}^l3
        │     │ L_FakeH_2[c] ← (K̃_1, K̃_2)
        │     │ K̃ ← (K̃_1, K̃_2)
        │     │ K ← K̃
        │     └ return K
        │ else
        │     │ Z ← c^sk
        │     │ K_1 ← H_2(Z, "CTAP2 HMAC KEY")
        │     │ K_2 ← H_2(Z, "CTAP2 AES KEY")
        │     │ K ← (K_1, K_2)
        └     └ return K
    if (_, pk) ∈ L_hy[hy : n_TokenPK[ then
        │ Z ← c^sk
        │ K_1 ← H_2(Z, "CTAP2 HMAC KEY")
        │ K_2 ← H_2(Z, "CTAP2 AES KEY")
        │ K ← (K_1, K_2)
        └ return K
```

**Fig. 40.** obtainSharedSecret-T on the left and PIN/UV Auth Protocol encapsulate and decapsulate functions on the right, simulated by $\mathcal{B}_1^i$.

$$\mathsf{Pr}_{0.i}[\mathsf{bad}] \geq \; \mid \mathsf{Pr}_{0.(i-1)} - \mathsf{Pr}_{0.i} \mid, \forall i \in [1, n_{\mathsf{TokenPK}}] \tag{25}$$

$$\mathsf{Adv}_{\mathsf{ECDH}}^{\mathsf{sCDH}}(\mathcal{B}_1^i) \geq \; \mid \mathsf{Pr}_{0.(i-1)} - \mathsf{Pr}_{0.i} \mid, \forall i \in [1, n_{\mathsf{TokenPK}}] \tag{26}$$

$$\sum_{i=1}^{n_{\mathsf{TokenPK}}} \mathsf{Adv}_{\mathsf{ECDH}}^{\mathsf{sCDH}}(\mathcal{B}_1^i) \geq \sum_{i=1}^{n_{\mathsf{TokenPK}}} \mid \mathsf{Pr}_{0.(i-1)} - \mathsf{Pr}_{0.i} \mid \tag{27}$$

$$\sum_{i=1}^{n_{\mathsf{TokenPK}}} \mathsf{Adv}_{\mathsf{ECDH}}^{\mathsf{sCDH}}(\mathcal{B}_1^i) \geq \; \mid \mathsf{Pr}_{0.0} - \mathsf{Pr}_{0.n_{\mathsf{TokenPK}}} \mid \tag{28}$$

Let $\mathcal{B}_1$ be the adversary with the highest advantage against the sCDH game out of all adversaries $\mathcal{B}_1^i$, $\forall i \in [1, n_{\mathsf{TokenPK}}]$. Then, it is always true that:

$$n_{\mathsf{TokenPK}} \, \mathsf{Adv}_{\mathsf{ECDH}}^{\mathsf{sCDH}}(\mathcal{B}_1) \geq \; \mid \mathsf{Pr}_{0.0} - \mathsf{Pr}_{0.n_{\mathsf{TokenPK}}} \mid \tag{29}$$

Since Game $0.0$ corresponds to Game 0 and Game $0.n_{\mathsf{TokenPK}}$ corresponds to Game 1, we have:

$$n_{\mathsf{TokenPK}} \, \mathsf{Adv}_{\mathsf{ECDH}}^{\mathsf{sCDH}}(\mathcal{B}_1) \geq \; \mid \mathsf{Pr}_0 - \mathsf{Pr}_1 \mid \tag{30}$$

Given that token ECDH public keys can only be given to $\mathcal{A}$ in Setup, Execute or Send-Bind-T queries, we know $n_{\text{TokenPK}} \leq (q_S + q_E + q_{\text{Send}})$. Therefore:

$$\left(q_S + q_E + q_{\text{Send}}\right) \, \mathsf{Adv}^{\text{sCDH}}_{\text{ECDH}}(\mathcal{B}_1) \geq \mid \mathsf{Pr}_0 - \mathsf{Pr}_1 \mid \tag{31}$$

## Game 2

In this game, the pinHash is no longer saved in the token's state, being saved instead on a new structure $\mathcal{L}_{\text{TokenSetup}}$ which is not available to $\mathcal{A}$.

---

Game2($\mathcal{A}$):
  $\mathcal{L}_{\text{authC}}, \mathcal{L}_{\text{authT}} \leftarrow \emptyset, \mathcal{L}_{\text{valid}} \leftarrow \emptyset, \mathcal{L}_{\text{corrupt}} \leftarrow \emptyset$
  $\mathcal{L}_{\text{sCDH}}, \mathcal{L}_{\mathcal{H}_2} \leftarrow \emptyset$
  $\mathcal{L}_{\text{TokenSetup}} \leftarrow \emptyset$
  win-SUF-t $\leftarrow 0$
  () $\xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(1^\lambda)$
  **return** win-SUF-t

---

**Fig. 41.** Game 2.

---

setPIN-T($\pi^i_T$, puvProtocol, $c, c_p, t_p$):
  **if** puvProtocol $\notin \mathsf{st}_T$.puvProtocolList $\vee$
  $\mathcal{L}_{\text{TokenSetup}}[T] \neq \perp$ **then return** $\perp$
  $K \leftarrow \mathsf{st}_T$.puvProtocol.decapsulate($c$)
  **if** $K = \perp \vee \mathsf{st}_T$.puvProtocol.verify($K, c_p, t_p$) = false
  **then return** $\perp$
  pin $\leftarrow \mathsf{st}_T$.puvProtocol.decrypt($K, c_p$)
  **if** pin $\notin \mathcal{P}$ **then return** $\perp$
  $\mathcal{L}_{\text{TokenSetup}}[T] \leftarrow \mathsf{H}(\text{pin})$
  $\mathsf{st}_T$.pinRetries $\leftarrow$ pinRetriesMax
  **return** accepted

obtainPinUvAuthToken-T($\pi^i_T$, puvProtocol, $c, c_{ph}$):
  **if** puvProtocol $\notin \mathsf{st}_T$.puvProtocolList $\vee \mathsf{st}_T$.pinRetries $= 0$ **then**
    $\quad$ **return** ($\perp$, false)
  $K \leftarrow \mathsf{st}_T$.puvProtocol.decapsulate($c$)
  **if** $K = \perp$ **then**
    $\quad$ **return** ($\perp$, false)
  $\mathsf{st}_T$.pinRetries $\leftarrow$ pinRetries $- 1$
  pinHash $\leftarrow \mathsf{st}_T$.puvProtocol.decrypt($K, c_{ph}$)
  pinHash$_T \leftarrow \mathcal{L}_{\text{TokenSetup}}[T]$
  **if** pinHash $\neq$ pinHash$_T$ **then**
    $\quad \mathsf{st}_T$.puvProtocol.regenerate()
    $\quad \mathsf{st}_T.m \leftarrow \mathsf{st}_T.m - 1$
    $\quad$ **if** $\mathsf{st}_T.m = 0$ **then**
      $\quad\quad$ authPowerUp-T($\mathsf{st}_T$)
      $\quad\quad$ **return** ($\perp$, true)
    $\quad$ **else**
      $\quad\quad$ **return** ($\perp$, false)
  $\mathsf{st}_T.m \leftarrow 3, \mathsf{st}_T$.pinRetries $\leftarrow$ pinRetriesMax
  **for all** puvProtocol' $\in \mathsf{st}_T$.puvProtocolList **do**
    $\quad \mathsf{st}_T$.puvProtocol'.resetPuvToken()
  $\pi^i_T$.bs $\leftarrow \pi^i_T$.puvProtocol.pt
  $c_{pt} \xleftarrow{\$} \mathsf{st}_T$.puvProtocol.encrypt($K, \pi^i_T$.bs)
  $\pi^i_T$.st$_{\text{exe}} \leftarrow$ bindDone
  $\pi^i_T$.canValidate $\leftarrow$ true
  $\pi^i_T$.sid $\leftarrow \pi^i_T$.sid $\|$ puvProtocol $\| c \| c_{ph} \| c_{pt} \|$ false
  **return** ($c_{pt}$, false)

---

**Fig. 42.** Modified CTAP 2.1+ setPIN-T and obtainPinUvAuthToken-T functions - Game 2.

**Advantage measure** Since this model does not allow tokens to be corrupted to reveal their state $\mathsf{st}_T$, Game 2 is functionally identical to Game 1, and we have:

$$\mathsf{Pr}_1 = \mathsf{Pr}_2 \tag{32}$$

## Game 3

In this game, $\mathcal{A}$ loses by causing a bad event **bad** if and only if the challenger samples two identical **ECDH** public keys, in either a token session or a client session. Both client and token sessions will sample a fresh **ECDH** key pair through the **regenerate** function.

```
Game3(A):
    L_authC, L_authT ← ∅, L_valid ← ∅, L_corrupt ← ∅
    L_sCDH, L_H₂ ← ∅
    L_TokenSetup ← ∅
    L_pk ← ∅
    bad ← false
    win-SUF-t ← 0
    () ⟵$ A^O(1^λ)
    return win-SUF-t
```

```
regenerate ():
    (pk, sk) ⟵$ ECDH.KG()
    if pk ∈ L_pk then
        bad ← true
        abort
    L_pk ← L_pk ∪ {pk}
```

**Fig. 43.** Game 3. The flag **bad** in red is only used in this game, to signal when a collision happens.

**Fig. 44.** Modifications **regenerate** - Game 3.

**Advantage measure** Each client-side **ECDH** public key is generated either on **Setup** or **Execute** queries, in the **obtainSharedSecret-C-start** function. Since each query always generates an **ECDH** public key, we have at most $q_S + q_E$ client **ECDH** public keys during the experiment.

Each token-side **ECDH** public key can be generated either during the creation of the token with the **NewT** oracle, by rebooting the token via the **Reboot** oracle, or during the **Send-Bind-T** oracle, in the **obtainPinUvAuthToken-T** function, whenever the **pinHash** verification fails. Since each **NewT** or **Reboot** query generates a new **ECDH** key pair every time, and each **Send-Bind-T** generates at most 2 **ECDH** key pairs (recall that **obtainPinUvAuthToken-T** regenerates whenever the **pinHash** verification fails but also when $\mathsf{st}_T.m$ reaches 0, which can happen during the same execution), we have at most $q_{\mathsf{NT}} + q_{\mathsf{R}} + 2q_{\mathsf{Send}}$ token **ECDH** public keys during the experiment.

Let $n_{pk} = q_S + q_E + q_{\mathsf{NT}} + q_{\mathsf{R}} + 2q_{\mathsf{Send}}$ be the total number of **ECDH** public keys generated through these oracles.

Considering the maximum probability of a collision between two keys as $1/q$, where $q$ is the prime order of the **ECDH** group used by the challenger throughout the experiment, and $n_{pk}^2/2$ as an upper bound for the amount of **ECDH** public keys pairs that can exist, we have:

$$\mathsf{Pr}_3[\mathsf{bad}] \leq n_{pk}^2/(2q) \tag{33}$$

Since games 2 and 3 are identical until **bad**, by the difference lemma, we have:

$$\mathsf{Pr}_3[\mathsf{bad}] \geq |\,\mathsf{Pr}_2 - \mathsf{Pr}_3\,| \tag{34}$$

And, therefore:

$$n_{pk}^2/(2q) \geq |\,\mathsf{Pr}_2 - \mathsf{Pr}_3\,| \tag{35}$$

Finally, from the definition of $n_{pk}$, we have:

$$\frac{(q_{\mathsf{S}} + q_{\mathsf{E}} + q_{\mathsf{NT}} + q_{\mathsf{R}} + 2q_{\mathsf{Send}})^2}{2q} \geq \mid \mathsf{Pr}_2 - \mathsf{Pr}_3 \mid \tag{36}$$

**Game 4**

In this game, the game aborts and $\mathcal{A}$ loses if when a collision occurs on some output of $\mathsf{H}$ when called by the challenger.

In order to easily represent the changes in code, and ensure every time a call to $\mathsf{H}$ is made the game checks for collisions, we create a wrapper function $\mathsf{H_w}$ which calls $\mathsf{H}$ and performs the collision check, and change every location in the entire experiment where $\mathsf{H}$ is called with $\mathsf{H_w}$.

---

Game4($\mathcal{A}$):
    $\mathcal{L}_{\mathsf{authC}}, \mathcal{L}_{\mathsf{authT}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{valid}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{corrupt}} \leftarrow \emptyset$
    $\mathcal{L}_{\mathsf{sCDH}}, \mathcal{L}_{\mathcal{H}_2} \leftarrow \emptyset$
    $\mathcal{L}_{\mathsf{TokenSetup}} \leftarrow \emptyset$
    $\mathcal{L}_{\mathsf{pk}} \leftarrow \emptyset$
    $\mathcal{L}_{\mathsf{hash}} \leftarrow \emptyset$
    $\mathsf{bad} \leftarrow \mathsf{false}$
    $\mathsf{win\text{-}SUF\text{-}t} \leftarrow 0$
    $() \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(1^\lambda)$
    **return** $\mathsf{win\text{-}SUF\text{-}t}$

**Fig. 45.** Game 4. The flag $\mathsf{bad}$ in red is only used in this game, to signal when a $\mathsf{pinHash}$ collision happens.

---

$\mathsf{H_w}$ (pin):
    $\mathsf{pinHash} \leftarrow \mathsf{H(pin)}$
    **for all** $(\mathsf{pin'}, \mathsf{pinHash'}) \in \mathcal{L}_{\mathsf{hash}}$ **do**
        **if** $\mathsf{pinHash'} = \mathsf{pinHash} \wedge \mathsf{pin'} \neq \mathsf{pin}$ **then**
            $\mathsf{bad} \leftarrow \mathsf{true}$
            **abort**
    $\mathcal{L}_{\mathsf{hash}}[\mathsf{pin}] \leftarrow \mathsf{pinHash}$
    **return** $\mathsf{pinHash}$

**Fig. 46.** New wrapper function $\mathsf{H_w}$ which calls $\mathsf{H}$ and aborts if a collision is found - Game 4.

---

**obtainPinUvAuthToken-C-start**($\pi_C^j$, pin):
    $\mathsf{pinHash} \leftarrow \mathsf{H_w(pin)}$
    $c_{ph} \xleftarrow{\$} \pi_C^j.\mathsf{selectedPuvProtocol.encrypt}(\pi_C^j.\mathsf{K}, \mathsf{pinHash})$
    $\pi_C^j.\mathsf{st}_{\mathsf{exe}} \leftarrow \mathsf{bindStart}$
    $\pi_C^j.\mathsf{sid} \leftarrow \pi_C^j.\mathsf{sid} \parallel c_{ph}$
    **return** $c_{ph}$

**setPIN-T**($\pi_T^i$, puvProtocol, $c, c_p, t_p$):
    **if** $\mathsf{puvProtocol} \notin \mathsf{st}_T.\mathsf{puvProtocolList} \vee$
    $\mathcal{L}_{\mathsf{TokenSetup}}[T] \neq \perp$ **then return** $\perp$
    $K \leftarrow \mathsf{st}_T.\mathsf{puvProtocol.decapsulate}(c)$
    **if** $K = \perp \vee \mathsf{st}_T.\mathsf{puvProtocol.verify}(K, c_p, t_p) = \mathsf{false}$
    **then return** $\perp$
    $\mathsf{pin} \leftarrow \mathsf{st}_T.\mathsf{puvProtocol.decrypt}(K, c_p)$
    **if** $\mathsf{pin} \notin \mathcal{P}$ **then return** $\perp$
    $\mathcal{L}_{\mathsf{TokenSetup}}[T] \leftarrow \mathsf{H_w(pin)}$
    $\mathsf{st}_T.\mathsf{pinRetries} \leftarrow \mathsf{pinRetriesMax}$
    **return** $\mathsf{accepted}$

**Fig. 47.** Modified CTAP 2.1+ functions obtainPinUvAuthToken-C-start and setPIN-T - Game 4.

---

**Advantage measure** Because Games 3 and 4 are identical until $\mathsf{bad}$, through the difference lemma, we have:

$$\mid \mathsf{Pr}_3 - \mathsf{Pr}_4 \mid \leq \mathsf{Pr}_4[\mathsf{bad}] \tag{37}$$

Since $\mathsf{H}$ is assumed to be collision resistant, there exists an adversary $\mathcal{B}_4$ such that:

$$\mathsf{Pr}_4[\mathsf{bad}] \leq \mathsf{Adv}_{\mathsf{H}}^{\mathsf{coll}}(\mathcal{B}_4) \tag{38}$$

Therefore:

$$| \ \mathsf{Pr}_3 - \mathsf{Pr}_4 \ | \leq \mathsf{Adv}_{\mathsf{H}}^{\mathsf{coll}}(\mathcal{B}_4) \tag{39}$$

**Game 5**

In this game, we replace all pins encrypted by any client session and sent to any token session during Setup with a constant pin 0000. Additionally, on any token session that shares the same session key $\tilde{K}$ with a different token session of the same token involved in a previous Setup query, we replace the encryption of the pinToken $pt$ with the encryption of a uniform random value $\tilde{pt}$.

Game5($\mathcal{A}$):
  $\mathcal{L}_{\mathsf{authC}}, \mathcal{L}_{\mathsf{authT}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{valid}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{corrupt}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{sCDH}}, \mathcal{L}_{\mathcal{H}_2} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{TokenSetup}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{pk}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{hash}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{pin}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{setupKeys}} \leftarrow \emptyset$
  win-SUF-t $\leftarrow 0$
  $() \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(1^\lambda)$
  **return** win-SUF-t

Setup $(T, i, C, j, U)$:
  $\mathsf{pin}_U \leftarrow \mathcal{L}_{\mathsf{valid}}[U]$
  **if** $\mathsf{st}_T = \perp$ **or** $\pi_T^i \neq \perp$ **or** $\pi_C^j \neq \perp$ **or** $\mathsf{pin}_U = \perp$ **then**
    $\llcorner$ **return** $\perp$
  $\pi_T^i \leftarrow \mathsf{st}_T$
  $\mathcal{L}_{\mathsf{pin}}[\pi_T^i] \leftarrow \mathsf{pin}_U$
  trans $\xleftarrow{\$} \mathsf{Setup}(\pi_T^i, \pi_C^j, \mathsf{pin}_U)$
  $\pi_T^i.\mathsf{isValid}, \pi_C^j.\mathsf{isValid} \leftarrow \mathsf{false}$
  $\mathsf{st}_T.\mathsf{user} \leftarrow U$
  **return** trans

**Fig. 48.** Game 5 and modified Setup oracle.

setPIN-C($\pi_C^j, \mathsf{pin}_U$):
  **if** $\mathsf{pin}_U \notin \mathcal{P}$ **then**
    $\llcorner$ **return** $\perp$
  $c_p \xleftarrow{\$} \pi_C^j.\mathsf{selectedPuvProtocol}.\mathsf{encrypt}(\pi_C^j.\mathsf{K}, 0000)$
  $t_p \xleftarrow{\$} \pi_C^j.\mathsf{selectedPuvProtocol}.\mathsf{authenticate}(\pi_C^j.\mathsf{K}, c_p)$
  **return** $(c_p, t_p)$

setPIN-T($\pi_T^i, \mathsf{puvProtocol}, c, c_p, t_p$):
  **if** $\mathsf{puvProtocol} \notin \mathsf{st}_T.\mathsf{puvProtocolList} \vee$
    $\mathcal{L}_{\mathsf{TokenSetup}}[T] \neq \perp$ **then return** $\perp$
  $\tilde{K} \leftarrow \mathsf{st}_T.\mathsf{puvProtocol}.\mathsf{decapsulate}(c)$
  **if** $\tilde{K} = \perp \vee \mathsf{st}_T.\mathsf{puvProtocol}.\mathsf{verify}(\tilde{K}, c_p, t_p) = \mathsf{false}$
  **then return** $\perp$
  $\mathsf{pin} \leftarrow \mathcal{L}_{\mathsf{pin}}[\pi_T^i]$
  **if** $\mathsf{pin} \notin \mathcal{P}$ **then return** $\perp$
  $\mathcal{L}_{\mathsf{setupKeys}}[T] \leftarrow (\mathsf{st}_T.\mathsf{puvProtocol}.pk, c)$
  $\mathcal{L}_{\mathsf{TokenSetup}}[T] \leftarrow \mathsf{H}_w(\mathsf{pin})$
  $\mathsf{st}_T.\mathsf{pinRetries} \leftarrow \mathsf{pinRetriesMax}$
  **return** accepted

**Fig. 49.** Modified CTAP 2.1+ functions setPIN-C and setPIN-T - Game 5.

We bound the advantage of $\mathcal{A}$ by reducing to the IND-1\$PA-LHPC security of SKE. The reason it does not suffice to reduce to the IND-1CPA security results from the observation that it is not guaranteed that the session key $\tilde{K}$ that encrypts the constant pin will not be used again to create more ciphertexts in the experiment. If $\mathcal{A}$ saves a client public key $c$ used to perform a Setup with a token session $\pi_T^i$ and then sends the same key $c$ to another session $\pi_T^{i'}$ of the same token during a Bind execution via the Send-Bind-T oracle, $\pi_T^{i'}$ might use the same session key $\tilde{K}_2$ to encrypt a pinToken, provided $\mathcal{A}$ can pass the pinHash verification step. This is possible because it is *not* guaranteed that a token $T$ will always generate a fresh ECDH key pair for every new key agreement.

```
obtainPinUvAuthToken-T($\pi_T^i$, puvProtocol, $c, c_{ph}$):
    if puvProtocol $\notin$ st$_T$.puvProtocolList $\lor$ st$_T$.pinRetries $= 0$ then
    └  return ($\bot$, false)
    $K \leftarrow$ st$_T$.puvProtocol.decapsulate($c$)
    if $K = \bot$ then
    └  return ($\bot$, false)
    st$_T$.pinRetries $\leftarrow$ pinRetries $- 1$
    pinHash $\leftarrow$ st$_T$.puvProtocol.decrypt($K, c_{ph}$)
    pinHash$_T \leftarrow \mathcal{L}_{\mathsf{TokenSetup}}[T]$
    if pinHash $\neq$ pinHash$_T$ then
    │   st$_T$.puvProtocol.regenerate()
    │   st$_T$.m $\leftarrow$ st$_T$.m $- 1$
    │   if st$_T$.m $= 0$ then
    │   │   authPowerUp-T(st$_T$)
    │   │   return ($\bot$, true)
    │   else
    │   └   return ($\bot$, false)
    st$_T$.m $\leftarrow 3$, st$_T$.pinRetries $\leftarrow$ pinRetriesMax
    for all puvProtocol' $\in$ st$_T$.puvProtocolList do
    └  st$_T$.puvProtocol'.resetPuvToken()
    $\pi_T^i$.bs $\leftarrow \pi_T^i$.puvProtocol.pt
    ▷ If this happens, then the ECDH key pair used in this execution was already used in
    a previous Setup query for token T.
    if (st$_T$.puvProtocol.pk, $c$) $= \mathcal{L}_{\mathsf{setupKeys}}[T]$ then
    │   $\tilde{pt} \xleftarrow{\$} \{0,1\}^{len(\pi_T^i.bs)}$
    │   $c_{pt} \xleftarrow{\$}$ st$_T$.puvProtocol.encrypt($K, \tilde{pt}$)
    else
    └  $c_{pt} \xleftarrow{\$}$ st$_T$.puvProtocol.encrypt($K, \pi_T^i$.bs)
    $\pi_T^i$.st$_{\mathsf{exe}} \leftarrow$ bindDone
    $\pi_T^i$.canValidate $\leftarrow$ true
    $\pi_T^i$.sid $\leftarrow \pi_T^i$.sid $||$ puvProtocol $|| c || c_{ph} || c_{pt} ||$ false
    return ($c_{pt}$, false)
```

**Fig. 50.** Modified CTAP 2.1+ function obtainPinUvAuthToken-T - Game 5.

In order to know, during the execution of obtainPinUvAuthToken-T, if the token session $\pi_T^i$ is reusing the same session key used in a previous Setup query, we create a structure $\mathcal{L}_{\mathsf{setupKeys}}$ which, during Setup, will fix the ECDH client and token public keys on token $T$. Later, if the ECDH public keys used in obtainPinUvAuthToken-T are the same for the given token $T$, we know it will be reusing the same session key, and thus we must replace the pinToken being encrypted.

**Hybrid Argument** To bound $\mathcal{A}$'s advantage against Game 5, we begin by creating $n_{\mathsf{setup}}$ hybrid games between Game 4 and Game 5, where $n_{\mathsf{setup}}$ is the maximum number of (valid) calls to the Setup oracle (valid refers to calls that actually result in executing the Setup between a client session and token session). Then, we replace the pin with the constant value 0000 and fix the ECDH public keys in $\mathcal{L}_{\mathsf{setupKeys}}$ only until the $i^{th}$ execution of Setup.

**Reduction to the IND-1\$PA-LHPC security of SKE** We now construct an adversary $\mathcal{B}_5^i$ against the IND-1\$PA-LHPC security of SKE that simulates game 4.$i$ to $\mathcal{A}$ perfectly, except when $\mathcal{A}$ performs the $i^{th}$ valid query to Setup. In this case, when executing setPIN-C, $\mathcal{B}_5^i$ sends pin and 0000 to $\mathcal{O}_{\mathsf{LR}}$ and receives a challenge ciphertext $c_p$, instead of encrypting it normally with the derived session key $\tilde{K}_2$. Then, when running setPIN-T on the corresponding token session, $\mathcal{B}_5^i$ flags the public key pair that was used during this $i^{th}$ Setup, to maintain consistency everytime the same public key pair appears later in the simulation, and sets the

```
Game4.i(A):                                          Setup (T, i, C, j, U):
    L_authC, L_authT ← ∅, L_valid ← ∅, L_corrupt ← ∅       pin_U ← L_valid[U]
    L_sCDH, L_H2 ← ∅                                        if st_T =⊥ or π^i_T ≠⊥ or π^j_C ≠⊥ or pin_U =⊥ then
    L_TokenSetup ← ∅                                            ⌊ return ⊥
    L_pk ← ∅                                                π^i_T ← st_T
    L_hash ← ∅                                              count_setup ← count_setup + 1
    L_pin ← ∅, L_setupKeys ← ∅                              L_pin[π^i_T] ← pin_U
    hy ← i, count_setup ← 0                                 trans ⟵$ Setup(π^i_T, π^j_C, pin_U)
    win-SUF-t ← 0                                           π^i_T.isValid, π^j_C.isValid ← false
    () ⟵$ A^O(1^λ)                                          st_T.user ← U
    return win-SUF-t                                        return trans
```

**Fig. 51.** Hybrid game $4.i$.

```
setPIN-C(π^j_C, pin_U):                                     setPIN-T(π^i_T, puvProtocol, c, c_p, t_p):
    if pin_U ∉ P then                                          if puvProtocol ∉ st_T.puvProtocolList ∨
    ⌊ return ⊥                                                 L_TokenSetup[T] ≠⊥ then return ⊥
    if count_setup ≤ hy then                                   K ← st_T.puvProtocol.decapsulate(c)
    |   c_p ⟵$ π^j_C.selectedPuvProtocol.encrypt(π^j_C.K, 0000)    if K =⊥ ∨ st_T.puvProtocol.verify(K, c_p, t_p) = false
    |   t_p ⟵$ π^j_C.selectedPuvProtocol.authenticate(π^j_C.K, c_p)   then return ⊥
    |   return (c_p, t_p)                                      if count_setup ≤ hy then
    else                                                      |   pin ← L_pin[π^i_T]
    |   c_p ⟵$ π^j_C.selectedPuvProtocol.encrypt(π^j_C.K, pin_U)   else
    |   t_p ⟵$ π^j_C.selectedPuvProtocol.authenticate(π^j_C.K, c_p)  ⌊ pin ← st_T.puvProtocol.decrypt(K, c_p)
    ⌊   return (c_p, t_p)                                      if pin ∉ P then return ⊥
                                                              if count_setup ≤ hy then
                                                              ⌊ L_setupKeys[T] ← (st_T.puvProtocol.pk, c)
                                                              L_TokenSetup[T] ← H_w(pin)
                                                              st_T.pinRetries ← pinRetriesMax
                                                              return accepted
```

**Fig. 52.** Modified CTAP 2.1+ functions setPIN-C and setPIN-T for hybrid game $4.i$.

pinHash correspoding to the pin that was saved in $\mathcal{L}_{pin}$ in the begginning of the Setup oracle execution.

If $\mathcal{A}$ later queries Send-Bind-T to the target token session $\pi^i_T$ that was involved in the $i^{th}$ Setup query with the public key $c$ that was used in that Setup (which will be identifiable by the flag set before in setPIN-T), $\mathcal{B}^i_5$ must perform a query to its LHPC oracle $\mathcal{O}_{LHPC}$ to check if the correct pinHash is present in the received ciphertext. If it's not, then the pinHash verification fails. Otherwise, the verification succeeds, and $\mathcal{B}^i_5$ must now use the RAND oracle $\mathcal{O}_{RAND}$ to encrypt a random pinToken to send to $\mathcal{A}$.

**Advantage measure** The advantage of $\mathcal{B}^i_5$ against the IND-1\$PA-LHPC security of SKE is:

$$\mathsf{Adv}^{\mathsf{IND\text{-}1\$PA\text{-}LHPC}}_{\mathsf{SKE}}(\mathcal{B}^i_5) = |\operatorname{Pr}_{\mathsf{IND\text{-}1\$PA\text{-}LHPC}}[b' = 0 | b = 1] - \operatorname{Pr}_{\mathsf{IND\text{-}1\$PA\text{-}LHPC}}[b' = 0 | b = 0]| \quad (40)$$

When oracle $\mathcal{O}_{LR}$ encrypts $m_0$, which corresponds to the real pin, then $\mathcal{B}^i_5$ is simulating Game $4.(i-1)$ to $\mathcal{A}$. If instead it encrypts $m_1$, corresponding to the constant pin 0000, then it simulates Game $4.i$.

From the definition of $\mathcal{B}^i_5$, it follows that:

$$\mathsf{Adv}^{\mathsf{IND\text{-}1\$PA\text{-}LHPC}}_{\mathsf{SKE}}(\mathcal{B}^i_5) = |\operatorname{Pr}_{4.i}[\mathsf{win\text{-}SUF\text{-}t} = 1] - \operatorname{Pr}_{4.(i-1)}[\mathsf{win\text{-}SUF\text{-}t} = 1]|, \forall i \in [1, n_{setup}] \quad (41)$$

$\mathcal{B}_5^i(\mathcal{A}, \mathcal{O}_{\mathsf{LR}}, \mathcal{O}_{\mathsf{RAND}}, \mathcal{O}_{\mathsf{LHPC}})$:
  $\mathcal{L}_{\mathsf{authC}}, \mathcal{L}_{\mathsf{authT}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{valid}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{corrupt}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{sCDH}}, \mathcal{L}_{\mathcal{H}_2} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{TokenSetup}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{pk}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{hash}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{pin}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{setupKeys}} \leftarrow \emptyset$
  $\mathsf{hy} \leftarrow i, \mathsf{count}_{\mathsf{setup}} \leftarrow 0$
  $pk_T^i, pk_C^i \leftarrow \bot$
  $c_{\mathsf{chal}} \leftarrow \bot$
  win-SUF-t $\leftarrow 0$
  $b' \leftarrow \bot$
  $() \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(1^\lambda)$
  **if** win-SUF-t $= 1$ **then**
  |  **return** $b' \leftarrow 0$
  **else**
  |  **return** $b' \leftarrow 1$
  **return** $b'$

Setup $(T, i, C, j, U)$:
  **if** $\mathsf{st}_T = \bot$ **or** $\pi_T^i \neq \bot$ **or** $\pi_C^j \neq \bot$ **or** $\mathsf{pin}_U = \bot$ **then**
  |  **return** $\bot$
  $\pi_T^i \leftarrow \mathsf{st}_T$
  $\mathsf{count}_{\mathsf{setup}} \leftarrow \mathsf{count}_{\mathsf{setup}} + 1$
  $\mathcal{L}_{\mathsf{pin}}[\pi_T^i] \leftarrow \mathsf{pin}_U$
  $\mathsf{trans} \xleftarrow{\$} \mathsf{Setup}(\pi_T^i, \pi_C^j, \mathsf{pin}_U)$
  $\pi_T^i.\mathsf{isValid}, \pi_C^j.\mathsf{isValid} \leftarrow \mathsf{false}$
  $\mathsf{st}_T.\mathsf{user} \leftarrow U$
  **return** $\mathsf{trans}$

**Fig. 53.** Adversary $\mathcal{B}_5^i$'s simulation of game $4.i$.

---

**setPIN-C**$(\pi_C^j, \mathsf{pin}_U)$:
  **if** $\mathsf{pin}_U \notin \mathcal{P}$ **then**
  |  **return** $\bot$
  **if** $\mathsf{count}_{\mathsf{setup}} = \mathsf{hy}$ **then**
  |  $c_p \xleftarrow{\$} \mathcal{O}_{\mathsf{LR}}(\mathsf{pin}_U, 0000)$
  |  $c_{\mathsf{chal}} \leftarrow c_p$
  |  $t_p \xleftarrow{\$} \pi_C^j.\mathsf{selectedPuvProtocol}.\mathsf{authenticate}(\pi_C^j.\mathsf{K}, c_p)$
  **else if** $\mathsf{count}_{\mathsf{setup}} < \mathsf{hy}$ **then**
  |  $c_p \xleftarrow{\$} \pi_C^j.\mathsf{selectedPuvProtocol}.\mathsf{encrypt}(\pi_C^j.\mathsf{K}, 0000)$
  |  $t_p \xleftarrow{\$} \pi_C^j.\mathsf{selectedPuvProtocol}.\mathsf{authenticate}(\pi_C^j.\mathsf{K}, c_p)$
  **else**
  |  $c_p \xleftarrow{\$} \pi_C^j.\mathsf{selectedPuvProtocol}.\mathsf{encrypt}(\pi_C^j.\mathsf{K}, \mathsf{pin}_U)$
  |  $t_p \xleftarrow{\$} \pi_C^j.\mathsf{selectedPuvProtocol}.\mathsf{authenticate}(\pi_C^j.\mathsf{K}, c_p)$
  **return** $(c_p, t_p)$

**setPIN-T**$(\pi_T^i, \mathsf{puvProtocol}, c, c_p, t_p)$:
  **if** $\mathsf{puvProtocol} \notin \mathsf{st}_T.\mathsf{puvProtocolList} \vee$
  $\mathcal{L}_{\mathsf{TokenSetup}}[T] \neq \bot$ **then return** $\bot$
  $K \leftarrow \mathsf{st}_T.\mathsf{puvProtocol}.\mathsf{decapsulate}(c)$
  **if** $K = \bot \vee \mathsf{st}_T.\mathsf{puvProtocol}.\mathsf{verify}(K, c_p, t_p) = \mathsf{false}$
  **then return** $\bot$
  **if** $\mathsf{count}_{\mathsf{setup}} = \mathsf{hy}$ **then**
  |  $pk_T^i \leftarrow \pi_T^i.\mathsf{puvProtocol}.pk$
  |  $pk_C^i \leftarrow c$
  **if** $\mathsf{count}_{\mathsf{setup}} \leq \mathsf{hy}$ **then**
  |  $\mathsf{pin} \leftarrow \mathcal{L}_{\mathsf{pin}}[\pi_T^i]$
  **else**
  |  $\mathsf{pin} \leftarrow \mathsf{st}_T.\mathsf{puvProtocol}.\mathsf{decrypt}(K, c_p)$
  **if** $\mathsf{pin} \notin \mathcal{P}$ **then return** $\bot$
  **if** $\mathsf{count}_{\mathsf{setup}} \leq \mathsf{hy}$ **then**
  |  $\mathcal{L}_{\mathsf{setupKeys}}[T] \leftarrow (\mathsf{st}_T.\mathsf{puvProtocol}.pk, c)$
  $\mathcal{L}_{\mathsf{TokenSetup}}[T] \leftarrow \mathsf{H}_w(\mathsf{pin})$
  $\mathsf{st}_T.\mathsf{pinRetries} \leftarrow \mathsf{pinRetriesMax}$
  **return** accepted

**obtainPinUvAuthToken-T**$(\pi_T^i, \mathsf{puvProtocol}, c, c_{ph})$:
  **if** $\mathsf{puvProtocol} \notin \mathsf{st}_T.\mathsf{puvProtocolList} \vee \mathsf{st}_T.\mathsf{pinRetries} = 0$
  **then**
  |  **return** $(\bot, \mathsf{false})$
  $K \leftarrow \mathsf{st}_T.\mathsf{puvProtocol}.\mathsf{decapsulate}(c)$
  **if** $K = \bot$ **then**
  |  **return** $(\bot, \mathsf{false})$
  $\mathsf{st}_T.\mathsf{pinRetries} \leftarrow \mathsf{pinRetries} - 1$
  **if** $\pi_T^i.\mathsf{puvProtocol}.pk = pk_T^i \wedge c = pk_C^i$ **then**
  |  **if** $c_{ph} = c_{\mathsf{chal}}$ **then**
  |  |  $\mathsf{pin} \leftarrow \mathcal{L}_{\mathsf{valid}}[\mathsf{st}_T.\mathsf{user}]$
  |  |  **if** $\mathsf{H}(\mathsf{pin}) = \mathsf{pin}$ **then**
  |  |  |  $\mathsf{pinHash} \leftarrow \mathsf{H}(\mathsf{pin})$
  |  |  **else**
  |  |  |  $\mathsf{pinHash} \leftarrow \bot$
  |  **else if** $\mathcal{O}_{\mathsf{LHPC}}(c_{ph}) = \mathsf{true}$ **then**
  |  |  $\mathsf{pinHash} \leftarrow \mathcal{L}_{\mathsf{TokenSetup}}[T]$
  |  **else**
  |  |  $\mathsf{pinHash} \leftarrow \bot$
  **else**
  |  $\mathsf{pinHash} \leftarrow \mathsf{st}_T.\mathsf{puvProtocol}.\mathsf{decrypt}(K, c_{ph})$
  $\mathsf{pinHash}_T \leftarrow \mathcal{L}_{\mathsf{TokenSetup}}[T]$
  **if** $\mathsf{pinHash} \neq \mathsf{pinHash}_T$ **then**
  |  $\mathsf{st}_T.\mathsf{puvProtocol}.\mathsf{regenerate}()$
  |  $\mathsf{st}_T.m \leftarrow \mathsf{st}_T.m - 1$
  |  **if** $\mathsf{st}_T.m = 0$ **then**
  |  |  $\mathsf{authPowerUp\text{-}T}(\mathsf{st}_T)$
  |  |  **return** $(\bot, \mathsf{true})$
  |  **else**
  |  |  **return** $(\bot, \mathsf{false})$
  $\mathsf{st}_T.m \leftarrow 3, \mathsf{st}_T.\mathsf{pinRetries} \leftarrow \mathsf{pinRetriesMax}$
  **for all** $\mathsf{puvProtocol}' \in \mathsf{st}_T.\mathsf{puvProtocolList}$ **do**
  |  $\mathsf{st}_T.\mathsf{puvProtocol}'.\mathsf{resetPuvToken}()$
  $\pi_T^i.\mathsf{bs} \leftarrow \pi_T^i.\mathsf{puvProtocol}.pt$
  **if** $(\mathsf{st}_T.\mathsf{puvProtocol}.pk, c) = \mathcal{L}_{\mathsf{setupKeys}}[T]$ **then**
  |  **if** $\pi_T^i.\mathsf{puvProtocol}.pk = pk_T^i \wedge c = pk_C^i$ **then**
  |  |  $(\_,\_, c_{pt}) \leftarrow \mathcal{O}_{\mathsf{RAND}}(\mathsf{len}(\pi_C^j.\mathsf{bs}))$
  |  **else**
  |  |  $\tilde{pt} \xleftarrow{\$} \{0,1\}^{len(\pi_T^i.\mathsf{bs})}$
  |  |  $c_{pt} \xleftarrow{\$} \mathsf{st}_T.\mathsf{puvProtocol}.\mathsf{encrypt}(K, \tilde{pt})$
  **else**
  |  $c_{pt} \xleftarrow{\$} \mathsf{st}_T.\mathsf{puvProtocol}.\mathsf{encrypt}(K, \pi_T^i.\mathsf{bs})$
  $\pi_T^i.\mathsf{st}_{\mathsf{exe}} \leftarrow \mathsf{bindDone}$
  $\pi_T^i.\mathsf{canValidate} \leftarrow \mathsf{true}$
  $\pi_T^i.\mathsf{sid} \leftarrow \pi_T^i.\mathsf{sid} \parallel \mathsf{puvProtocol} \parallel c \parallel c_{ph} \parallel c_{pt} \parallel \mathsf{false}$
  **return** $(c_{pt}, \mathsf{false})$

**Fig. 54.** CTAP 2.1+ functions setPIN-C, setPIN-T and obtainPinUvAuthToken-T simulated by $\mathcal{B}_5^i$.

$$\mathsf{Adv}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LHPC}}(\mathcal{B}_5^i) = |\ \mathsf{Pr}_{4.i} - \mathsf{Pr}_{4.(i-1)}\ |, \forall i \in [1, n_{\mathsf{setup}}] \tag{42}$$

$$\sum_{i=1}^{n_{\mathsf{setup}}} \mathsf{Adv}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LHPC}}(\mathcal{B}_5^i) = \sum_{i=1}^{n_{\mathsf{setup}}} |\ \mathsf{Pr}_{4.i} - \mathsf{Pr}_{4.(i-1)}\ | \tag{43}$$

$$\sum_{i=1}^{n_{\mathsf{setup}}} \mathsf{Adv}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LHPC}}(\mathcal{B}_5^i) \geq |\ \mathsf{Pr}_{4.0} - \mathsf{Pr}_{4.n_{\mathsf{setup}}}\ | \tag{44}$$

Let $\mathcal{B}_5$ be the adversary with the highest advantage against the IND-1\$PA-LHPC game, $\forall i \in [1, n_{\mathsf{setup}}]$. Then, it is always true that:

$$n_{\mathsf{setup}}\ \mathsf{Adv}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LHPC}}(\mathcal{B}_5) \geq \sum_{i=1}^{n_{\mathsf{setup}}} \mathsf{Adv}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LHPC}}(\mathcal{B}_5^i) \tag{45}$$

And thus,

$$n_{\mathsf{setup}}\ \mathsf{Adv}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LHPC}}(\mathcal{B}_5) \geq |\ \mathsf{Pr}_{4.0} - \mathsf{Pr}_{4.n_{\mathsf{setup}}}\ | \tag{46}$$

Since Game 4.0 corresponds to Game 4 and Game $4.n_{\mathsf{setup}}$ corresponds to Game 5, we have:

$$n_{\mathsf{setup}}\ \mathsf{Adv}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LHPC}}(\mathcal{B}_5) \geq |\ \mathsf{Pr}_4 - \mathsf{Pr}_5\ | \tag{47}$$

And, since $n_{\mathsf{setup}}$ is the maximum amount of valid Setup queries made by $\mathcal{A}$, we have that $n_{\mathsf{setup}} \leq q_{\mathsf{S}}$. Therefore:

$$q_{\mathsf{S}}\ \mathsf{Adv}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LHPC}}(\mathcal{B}_5) \geq |\ \mathsf{Pr}_4 - \mathsf{Pr}_5\ | \tag{48}$$

**Game 6**

In this game, we replace all pinHashes encrypted by any client session in Execute with the hash of a constant value 0000 and replace all pinTokens encrypted by any token session that was not actively attacked by $\mathcal{A}$ with a random pinToken instead. We skip any pinHash verification for any passive attempt (when $c$ and $c_{ph}$ were delivered to the token via an Execute query or when $\mathcal{A}$ delivers some $c$ and $c_{ph}$ to a token via the Send-Bind-T oracle that were output by a client session in a previous Execute, as long as the token is using the same public key). We then bound the advantage of $\mathcal{A}$ by reducing to the IND-1\$PA-LPC security of SKE.

We create a new structure $\mathcal{L}_{\mathsf{ct}}$ which saves important information from the client session, namely its public key $c$ and ciphertext $c_{ph}$ containing the constant pinHash. Then, the game behaves differently depending on which oracle called obtainPinUvAuthToken-T.

If $c$ and $c_{ph}$ are transmitted to a token session $\pi_T^i$ during an Execute query by a client session $\pi_C^j$, which is guaranteed to be passive, then the pinHash verification is skipped, and

Game6 ($\mathcal{A}$):
    $\mathcal{L}_{\mathsf{authC}}, \mathcal{L}_{\mathsf{authT}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{valid}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{corrupt}} \leftarrow \emptyset$
    $\mathcal{L}_{\mathsf{sCDH}}, \mathcal{L}_{\mathcal{H}_2} \leftarrow \emptyset$
    $\mathcal{L}_{\mathsf{TokenSetup}} \leftarrow \emptyset$
    $\mathcal{L}_{\mathsf{pk}} \leftarrow \emptyset$
    $\mathcal{L}_{\mathsf{hash}} \leftarrow \emptyset$
    $\mathcal{L}_{\mathsf{pin}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{setupKeys}} \leftarrow \emptyset$
    $\mathcal{L}_{\mathsf{ct}} \leftarrow \emptyset$
    win-SUF-t $\leftarrow 0$
    $() \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(1^\lambda)$
    **return** win-SUF-t

Execute $(T, i, C, j)$:
    $\mathsf{pin}_U \leftarrow \mathcal{L}_{\mathsf{valid}}[\mathsf{st}_T.\mathsf{user}]$
    **if** $\mathsf{st}_T = \perp$ **or** $\pi_T^i \neq \perp$ **or** $\pi_C^j \neq \perp$ **or** $\mathsf{pin}_U = \perp$ **then**
        **return** $\perp$
    $\pi_T^i \leftarrow \mathsf{st}_T$
    trans, $m_C, m_T \leftarrow \perp$
    ▷ *Each $\pi_C^j$ and $\pi_T^i$ can be used only once in each Execute, Send-Bind-T and Send-Bind-C.*
    $\mathcal{L}_{\mathsf{ct}}[\pi_C^j, \pi_T^i] \leftarrow (\perp, \perp, \perp)$
    **while** $\pi_C^j.\mathsf{st}_{\mathsf{exe}} \neq \mathsf{bindDone}$ **do**
        $m_T \xleftarrow{\$} \mathsf{Bind\text{-}T}(\pi_T^i, m_C)$
        $m_C \xleftarrow{\$} \mathsf{Bind\text{-}C}(\pi_C^j, U, m_T)$
        trans $\leftarrow$ trans $|| m_T || m_C$
    **for all** $i' \neq i$ **and** $\pi_T^{i'} \neq \perp$ **do**
        $\pi_T^{i'}.\mathsf{isValid} \leftarrow \mathsf{false}$
    **return** trans

obtainPinUvAuthToken-C-start$(\pi_C^j, \mathsf{pin})$:
    pinHash $\leftarrow H_w(0000)$
    $c_{ph} \xleftarrow{\$} \pi_C^j.\mathsf{puvProtocol}.\mathsf{encrypt}(\pi_C^j.\mathsf{K}, \mathsf{pinHash})$
    $\pi_C^j.\mathsf{st}_{\mathsf{exe}} \leftarrow \mathsf{bindStart}$
    $\pi_C^j.\mathsf{sid} \leftarrow \pi_C^j.\mathsf{sid} || c_{ph}$
    $\mathcal{L}_{\mathsf{ct}}[\pi_C^j, \_] \leftarrow (\pi_C^j.\mathsf{puvProtocol}.pk, c_{ph}, \perp)$
    **return** $c_{ph}$

obtainPinUvAuthToken-C-end$(\pi_C^j, c_{pt})$:
    $(\_, \_, pt) \leftarrow \mathcal{L}_{\mathsf{ct}}[\pi_C^j, \_]$
    $\pi_C^j.\mathsf{bs} \leftarrow pt$
    $\pi_C^j.\mathsf{st}_{\mathsf{exe}} \leftarrow \mathsf{bindDone}$
    $\pi_C^j.\mathsf{sid} \leftarrow \pi_C^j.\mathsf{sid} || c_{pt} || \mathsf{false}$
    **return**

obtainPinUvAuthToken-T$(\pi_T^i, \mathsf{puvProtocol}, c, c_{ph})$:
    **if** puvProtocol $\notin \mathsf{st}_T.\mathsf{puvProtocolList} \vee \mathsf{st}_T.\mathsf{pinRetries} = 0$ **then**
        **return** $(\perp, \mathsf{false})$
    $K \leftarrow \mathsf{st}_T.\mathsf{puvProtocol}.\mathsf{decapsulate}(c)$
    **if** $K = \perp$ **then**
        **return** $(\perp, \mathsf{false})$
    $\mathsf{st}_T.\mathsf{pinRetries} \leftarrow \mathsf{pinRetries} - 1$
    $\mathsf{active}_{\mathsf{key}}, \mathsf{active}_{c_{ph}} \leftarrow \mathsf{true}$
    **for all** $(\pi_{C'}^{j'}, \pi_{T'}^{i'}, c', c'_{ph}, \_) \in \mathcal{L}_{\mathsf{ct}}$ s.t. $T' = T \wedge \pi_{T'}^{i'}.\mathsf{puvProtocol}.pk = \pi_T^i.\mathsf{puvProtocol}.pk$ **do**
        **if** $c' = c$ **then**
            $\mathsf{active}_{\mathsf{key}} \leftarrow \mathsf{false}$
        **if** $c'_{ph} = c_{ph} \wedge c' = c$ **then**
            $\mathsf{active}_{c_{ph}} \leftarrow \mathsf{false}$
            break
    $\mathsf{verifyPin} \leftarrow \mathsf{active}_{\mathsf{key}} \vee \mathsf{active}_{c_{ph}}$
    **if** $\mathsf{verifyPin} = \mathsf{true}$ **then**
        $\mathsf{pinHash} \leftarrow \mathsf{st}_T.\mathsf{puvProtocol}.\mathsf{decrypt}(K, c_{ph})$
        $\mathsf{pinHash}_T \leftarrow \mathcal{L}_{\mathsf{TokenSetup}}[T]$
        **if** $\mathsf{pinHash} \neq \mathsf{pinHash}_T$ **then**
            $\mathsf{st}_T.\mathsf{puvProtocol}.\mathsf{regenerate}()$
            $\mathsf{st}_T.m \leftarrow \mathsf{st}_T.m - 1$
            **if** $\mathsf{st}_T.m = 0$ **then**
                $\mathsf{authPowerUp\text{-}T}(\mathsf{st}_T)$
                **return** $(\perp, \mathsf{true})$
            **else**
                **return** $(\perp, \mathsf{false})$
    $\mathsf{st}_T.m \leftarrow 3, \mathsf{st}_T.\mathsf{pinRetries} \leftarrow \mathsf{pinRetriesMax}$
    **for all** puvProtocol' $\in \mathsf{st}_T.\mathsf{puvProtocolList}$ **do**
        $\mathsf{st}_T.\mathsf{puvProtocol}'.\mathsf{resetPuvToken}()$
    $\pi_T^i.\mathsf{bs} \leftarrow \pi_T^i.\mathsf{puvProtocol}.pt$
    **if** $\mathsf{active}_{\mathsf{key}} = \mathsf{true} \wedge (\mathsf{st}_T.\mathsf{puvProtocol}.pk, c) \neq \mathcal{L}_{\mathsf{setupKeys}}[T]$ **then**
        $c_{pt} \xleftarrow{\$} \mathsf{st}_T.\mathsf{puvProtocol}.\mathsf{encrypt}(K, \pi_T^i.\mathsf{bs})$
    **else**
        $\tilde{pt} \xleftarrow{\$} \{0,1\}^{len(\pi_T^i.\mathsf{bs})}$
        $c_{pt} \xleftarrow{\$} \mathsf{st}_T.\mathsf{puvProtocol}.\mathsf{encrypt}(K, \tilde{pt})$
        $\mathcal{L}_{\mathsf{ct}}[\_, \pi_T^i] \leftarrow (c', c'_{ph}, \pi_T^i.\mathsf{bs})$
    $\pi_T^i.\mathsf{st}_{\mathsf{exe}} \leftarrow \mathsf{bindDone}$
    $\pi_T^i.\mathsf{canValidate} \leftarrow \mathsf{true}$
    $\pi_T^i.\mathsf{sid} \leftarrow \pi_T^i.\mathsf{sid} || \mathsf{puvProtocol} || c || c_{ph} || c_{pt} || \mathsf{false}$
    **return** $(c_{pt}, \mathsf{false})$

**Fig. 55.** Game 6, modified Execute oracle and modified CTAP 2.1+ functions obtainPinUvAuthToken-C-start, obtainPinUvAuthToken-C-end and obtainPinUvAuthToken-T - Game 6.

a new random pinToken is encrypted instead of the real pinToken generated by $\pi_T^i$ that is set as the binding state $\pi_T^i.\mathsf{bs}$ and saved in $\mathcal{L}_{\mathsf{ct}}$. The ciphertext $c_{pt}$ is then sent to $\pi_C^j$, which simply discards it and sets the same binding state as $\pi_T^i$.

If $c$ and $c_{ph}$ are sent by $\mathcal{A}$ to a token session $\pi_T^i$ via a Send-Bind-T query, the received $c$ and $c_{ph}$ are treated differently in obtainPinUvAuthToken-T depending on their origin. If both $c$ and $c_{ph}$ were created by a client session $\pi_C^{j'}$ that completed a Bind with a different token session $\pi_T^{i'}$ via an Execute query, and $\pi_T^{i'}$ and $\pi_T^i$ share the same ECDH public key, then the experiment skips the pinHash verification (we say that $c$ and $c_{ph}$ are passive). Recall that any Bind session from the Execute oracle is passive, which means that client sessions involved in an Execute query for some token $T$ must have not only received $T$'s public key, but also be the only client sessions to have received that key (collisions between public keys were removed in a previous game, ruling out different tokens with the same public key). Any one of those client sessions, as a result, encrypts the same pin in the original experiment, in such a way that any of $T$'s sessions can always correctly decrypt into the pinHash if they derive the same session key $K$. Therefore, a token session $\pi_T^i$ can always determine that, as long as $\mathcal{A}$ delivers a $c$ and $c_{ph}$ that were created in a client session that received $\pi_T^i$'s ECDH public key, $c_{ph}$ contains the correct pinHash.

If $c$ has never appeared in an Execute query with token $T$ before, then $\pi_T^i$ must derive the session key normally, since it means that $\mathcal{A}$ has started a Bind session with $\pi_T^i$ without any client session being involved, and decrypt $c_{ph}$ normally. If $c$ has appeared in a client session $\pi_C^j$ before, and $\pi_T^i$'s ECDH public key is the same that $\pi_C^j$ received, but $c_{ph}$ is different, then it must be decrypted normally in order to verify the decrypted pinHash.

After the verification step, if $c$ is passive, a new random pinToken $\tilde{pt}$ is encrypted instead of the real pinToken $pt$ that was set as the binding state, resulting in a ciphertext $c_{pt}$. The resulting binding state is stored in $\mathcal{L}_{\mathsf{ct}}$, and $c_{pt}$ is sent to $\mathcal{A}$.

**Hybrid Argument** To bound $\mathcal{A}$'s advantage against Game 6, we begin by creating $n_{\mathsf{execute}}$ hybrid games, where $n_{\mathsf{execute}}$ is the maximum number of (valid) calls to the Execute oracle (valid refers to calls that actually result in executing the Execute between a client session and token session). We replace the hashed pin with the hash of the constant value 0000 in client sessions and stop pinHash verification and replace pinToken encryption with the encryption of a random value in token sessions only until the $i^{th}$ execution of Execute.

Additionally, we also skip verifications of the pinHash in any token session that receives passive $c$ and $c_{ph}$ that were involved in the first $i^{th}$ Execute queries, as long as the token session is reusing the same ECDH public key, and replace all pinTokens with random values in any token session that receives a passive $c$ and passes pinHash verification, as long as $c$ came from one of the first $i^{th}$ Execute queries and the token session is reusing the same ECDH public key.

A new temporary structure $\mathcal{L}_{\mathsf{executePk}}$ is created so that each client and token session can be tracked throughout the experiment, by linking each client and token ECDH public key pair with a specific Execute query. This allows us to maintain consistency with $\mathcal{A}$ whenever

it queries Send-Bind-T with an ECDH public key $c$ that was used by a previous client session during Execute.

**Reduction to the IND-1\$PA-LPC security of SKE** We now construct an adversary $\mathcal{B}_6^i$ against the IND-1\$PA-LPC security of SKE that simulates game $5.i$ to $\mathcal{A}$ perfectly, except when $\mathcal{A}$ performs the $i^{th}$ valid query to Execute. In this case, $\mathcal{B}_6^i$ sends $H(\text{pin})$ and $H(0000)$ to $\mathcal{O}_{LR}$ and receives a challenge ciphertext $c_{ph}$, instead of encrypting it normally with the derived session key $\tilde{K}_2$. When these are delivered to the token session $\pi_T^i$, verification always succeeds when performed during Execute. $\mathcal{B}_6^i$ calls $\mathcal{O}_{RAND}$ to produce ciphertext $c_{pt}$ and saves $pt_0$ as the binding state. Since this was all during Execute, $c_{ph}$ is delivered to the client session $\pi_C^j$, which simply keeps the correct binding state $pt_0$.

If $\mathcal{A}$ calls Send-Bind-T on a token session $\pi_T^i$ such that the provided public key $c$ and $\pi_T^i$'s public key $pk$ were involved in the $i^{th}$ Execute query, but $c_{ph}$ was not the output ciphertext ($c_{ph}$ is not the challenge ciphertext output by $\mathcal{O}_{LR}$), then $\mathcal{B}_6^i$ must use the $\mathcal{O}_{LPC}$ oracle to check if pinHash is in the received ciphertext $c_{ph}$. In this situation, if the verification succeeds, then $\mathcal{B}_6^i$ still uses the $\mathcal{O}_{RAND}$ oracle to encrypt a random pinToken, and delivers $c_{pt}$ to $\mathcal{A}$.

**Advantage measure** The advantage of $\mathcal{B}_6^i$ against the IND-1\$PA-LPC security of SKE during game $5.i$ is:

$$\text{Adv}_{SKE}^{IND\text{-}1\$PA\text{-}LPC}(\mathcal{B}_6^i) = \mid \text{Pr}_{IND\text{-}1\$PA\text{-}LPC}[b'=0|b=1] - \text{Pr}_{IND\text{-}1\$PA\text{-}LPC}[b'=0|b=0] \mid \qquad (49)$$

When oracle $\mathcal{O}_{LR}$ encrypts $m_0$, which corresponds to $H(\text{pin})$, the oracle $\mathcal{O}_{RAND}$ encrypts $pt_0$, which is the pinToken being stored as the binding state in $\pi_T^i.\text{bs}$. Then, $\mathcal{B}_6^i$ is simulating Game $5.(i-1)$ to $\mathcal{A}$, where both $c_{ph}$ and $c_{pt}$ have the correct pinHash and pinToken respectively. If $\mathcal{O}_{LR}$ instead encrypts $m_1$, which is $H(0000)$, then $\mathcal{O}_{RAND}$ encrypts $pt_1$, and $\mathcal{B}_6^i$ is simulating Game $5.i$.

From the definition of $\mathcal{B}_6^i$, it follows that:

$$\text{Adv}_{SKE}^{IND\text{-}1\$PA\text{-}LPC}(\mathcal{B}_6^i) = \mid \text{Pr}_{5.i}[\text{win-SUF-t} = 1] - \text{Pr}_{5.(i-1)}[\text{win-SUF-t} = 1] \mid, \forall i \in [1, n_{\text{execute}}] \quad (50)$$

$$\text{Adv}_{SKE}^{IND\text{-}1\$PA\text{-}LPC}(\mathcal{B}_6^i) = \mid \text{Pr}_{5.i} - \text{Pr}_{5.(i-1)} \mid, \forall i \in [1, n_{\text{execute}}] \qquad (51)$$

$$\sum_{i=1}^{n_{\text{execute}}} \text{Adv}_{SKE}^{IND\text{-}1\$PA\text{-}LPC}(\mathcal{B}_6^i) = \sum_{i=1}^{n_{\text{execute}}} \mid \text{Pr}_{5.i} - \text{Pr}_{5.(i-1)} \mid \qquad (52)$$

$$\sum_{i=1}^{n_{\text{execute}}} \text{Adv}_{SKE}^{IND\text{-}1\$PA\text{-}LPC}(\mathcal{B}_6^i) \geq \mid \text{Pr}_{5.0} - \text{Pr}_{5.n_{\text{execute}}} \mid \qquad (53)$$

Let $\mathcal{B}_6$ be the adversary with the highest advantage against the IND-1\$PA-LPC game out of all adversaries $\mathcal{B}_6^i$, $\forall i \in [1, n_{\text{execute}}]$. Then, it is always true that:

**Game5.$i$ ($\mathcal{A}$):**

$\quad \mathcal{L}_{\mathsf{authC}}, \mathcal{L}_{\mathsf{authT}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{valid}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{corrupt}} \leftarrow \emptyset$
$\quad \mathcal{L}_{\mathsf{sCDH}}, \mathcal{L}_{\mathcal{H}_2} \leftarrow \emptyset$
$\quad \mathcal{L}_{\mathsf{TokenSetup}} \leftarrow \emptyset$
$\quad \mathcal{L}_{\mathsf{pk}} \leftarrow \emptyset$
$\quad \mathcal{L}_{\mathsf{hash}} \leftarrow \emptyset$
$\quad \mathcal{L}_{\mathsf{pin}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{setupKeys}} \leftarrow \emptyset$
$\quad \mathcal{L}_{\mathsf{ct}} \leftarrow \emptyset$
$\quad \mathsf{hy} \leftarrow i, \mathsf{count}_{\mathsf{execute}} \leftarrow 0, \mathcal{L}_{\mathsf{executePk}} \leftarrow \emptyset$
$\quad \pi_C^*, \pi_T^* \leftarrow \perp$
$\quad \mathsf{win\text{-}SUF\text{-}t} \leftarrow 0$
$\quad () \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(1^\lambda)$
$\quad \textbf{return } \mathsf{win\text{-}SUF\text{-}t}$

**Execute $(T, i, C, j)$:**

$\quad \mathsf{pin}_U \leftarrow \mathcal{L}_{\mathsf{valid}}[U]$
$\quad \textbf{if } \mathsf{st}_T = \perp \textbf{ or } \pi_T^i \neq \perp \textbf{ or } \pi_C^j \neq \perp \textbf{ or } \mathsf{pin}_U = \perp \textbf{ then}$
$\quad\quad \llcorner \ \textbf{return } \perp$
$\quad \pi_T^i \leftarrow \mathsf{st}_T$
$\quad \mathsf{trans}, m_C, m_T \leftarrow \perp$
$\quad \mathcal{L}_{\mathsf{ct}}[\pi_C^j, \pi_T^i] \leftarrow (\perp, \perp, \perp)$
$\quad \mathsf{count}_{\mathsf{execute}} \leftarrow \mathsf{count}_{\mathsf{execute}} + 1$
$\quad \pi_C^* \leftarrow \pi_C^j, \pi_T^* \leftarrow \pi_T^i$
$\quad \textbf{while } \pi_C^j.\mathsf{st}_{\mathsf{exe}} \neq \mathsf{bindDone} \textbf{ do}$
$\quad\quad | \quad m_T \xleftarrow{\$} \mathsf{Bind\text{-}T}(\pi_T^i, m_C)$
$\quad\quad | \quad m_C \xleftarrow{\$} \mathsf{Bind\text{-}C}(\pi_C^j, U, m_T)$
$\quad\quad | \quad \mathsf{trans} \leftarrow \mathsf{trans} \ || \ m_T \ || \ m_C$
$\quad \textbf{for all } i' \neq i \textbf{ and } \pi_T^{i'} \neq \perp \textbf{ do}$
$\quad\quad \llcorner \ \pi_T^{i'}.\mathsf{isValid} \leftarrow \mathsf{false}$
$\quad \textbf{return } \mathsf{trans}$

**obtainPinUvAuthToken-C-start$(\pi_C^j, \mathsf{pin})$:**

$\quad \textbf{if } \mathsf{count}_{\mathsf{execute}} \leq \mathsf{hy} \textbf{ then}$
$\quad\quad | \quad \mathsf{pinHash} \leftarrow \mathsf{H_w}(0000)$
$\quad \textbf{else}$
$\quad\quad \llcorner \ \mathsf{pinHash} \leftarrow \mathsf{H_w}(\mathsf{pin})$
$\quad pk_{\mathsf{client}} \leftarrow \pi_C^*.\mathsf{puvProtocol}.pk$
$\quad pk_{\mathsf{token}} \leftarrow \pi_T^*.\mathsf{puvProtocol}.pk$
$\quad \mathcal{L}_{\mathsf{executePk}}[pk_{\mathsf{client}}, pk_{\mathsf{token}}] \leftarrow \mathsf{count}_{\mathsf{execute}}$
$\quad c_{ph} \xleftarrow{\$} \pi_C^j.\mathsf{puvProtocol}.\mathsf{encrypt}(\pi_C^j.\mathsf{K}, \mathsf{pinHash})$
$\quad \pi_C^j.\mathsf{st}_{\mathsf{exe}} \leftarrow \mathsf{bindStart}$
$\quad \pi_C^j.\mathsf{sid} \leftarrow \pi_C^j.\mathsf{sid} \ || \ c_{ph}$
$\quad \mathcal{L}_{\mathsf{ct}}[\pi_C^j, \_] \leftarrow (\pi_C^j.\mathsf{puvProtocol}.pk, c_{ph}, \perp)$
$\quad \textbf{return } c_{ph}$

**obtainPinUvAuthToken-C-end$(\pi_C^j, c_{pt})$:**

$\quad \textbf{if } \mathsf{count}_{\mathsf{execute}} \leq \mathsf{hy} \textbf{ then}$
$\quad\quad | \quad (\_, \_, pt) \leftarrow \mathcal{L}_{\mathsf{ct}}[\pi_C^j, \_]$
$\quad\quad | \quad \pi_C^j.\mathsf{bs} \leftarrow pt$
$\quad \textbf{else}$
$\quad\quad \llcorner \ \pi_C^j.\mathsf{bs} \leftarrow \pi_C^j.\mathsf{puvProtocol}.\mathsf{decrypt}(\pi_C^j.\mathsf{K}, c_{pt})$
$\quad \pi_C^j.\mathsf{st}_{\mathsf{exe}} \leftarrow \mathsf{bindDone}$
$\quad \pi_C^j.\mathsf{sid} \leftarrow \pi_C^j.\mathsf{sid} \ || \ c_{pt} \ || \ \mathsf{false}$
$\quad \textbf{return}$

**obtainPinUvAuthToken-T$(\pi_T^i, \mathsf{puvProtocol}, c, c_{ph})$:**

$\quad \textbf{if } \mathsf{puvProtocol} \notin \mathsf{st}_T.\mathsf{puvProtocolList} \vee \mathsf{st}_T.\mathsf{pinRetries} = 0$
$\quad \textbf{then}$
$\quad\quad \llcorner \ \textbf{return } (\perp, \mathsf{false})$
$\quad K \leftarrow \mathsf{st}_T.\mathsf{puvProtocol}.\mathsf{decapsulate}(c)$
$\quad \textbf{if } K = \perp \textbf{ then}$
$\quad\quad \llcorner \ \textbf{return } (\perp, \mathsf{false})$
$\quad \mathsf{st}_T.\mathsf{pinRetries} \leftarrow \mathsf{pinRetries} - 1$
$\quad \mathsf{active}_{\mathsf{key}}, \mathsf{active}_{c_{ph}} \leftarrow \mathsf{true}$
$\quad \textbf{for all } (\pi_{C'}^{j'}, \pi_{T'}^{i'}, c', c'_{ph}, \_) \in \mathcal{L}_{\mathsf{ct}} \textbf{ s.t. } T' = T \wedge$
$\quad \pi_{T'}^{i'}.\mathsf{puvProtocol}.pk = \pi_T^i.\mathsf{puvProtocol}.pk \textbf{ do}$
$\quad\quad | \quad \textbf{if } c' = c \textbf{ then}$
$\quad\quad | \quad\quad \llcorner \ \mathsf{active}_{\mathsf{key}} \leftarrow \mathsf{false}$
$\quad\quad | \quad \textbf{if } c'_{ph} = c_{ph} \wedge c' = c \textbf{ then}$
$\quad\quad | \quad\quad | \quad \mathsf{active}_{c_{ph}} \leftarrow \mathsf{false}$
$\quad\quad | \quad\quad \llcorner \ \mathsf{break}$
$\quad \mathsf{verifyPin} \leftarrow \mathsf{active}_{\mathsf{key}} \vee \mathsf{active}_{c_{ph}}$
$\quad \mathsf{count}_{\mathsf{execute}} \leftarrow \mathcal{L}_{\mathsf{executePk}}[c, \pi_T^i.\mathsf{puvProtocol}.pk]$
$\quad \textbf{if } \mathsf{verifyPin} = \mathsf{true} \vee \mathsf{count}_{\mathsf{execute}} > \mathsf{hy} \textbf{ then}$
$\quad\quad | \quad \mathsf{pinHash} \leftarrow \mathsf{st}_T.\mathsf{puvProtocol}.\mathsf{decrypt}(K, c_{ph})$
$\quad\quad | \quad \mathsf{pinHash}_T \leftarrow \mathcal{L}_{\mathsf{TokenSetup}}[T]$
$\quad\quad | \quad \textbf{if } \mathsf{pinHash} \neq \mathsf{pinHash}_T \textbf{ then}$
$\quad\quad | \quad\quad | \quad \mathsf{st}_T.\mathsf{puvProtocol}.\mathsf{regenerate}()$
$\quad\quad | \quad\quad | \quad \mathsf{st}_T.m \leftarrow \mathsf{st}_T.m - 1$
$\quad\quad | \quad\quad | \quad \textbf{if } \mathsf{st}_T.m = 0 \textbf{ then}$
$\quad\quad | \quad\quad | \quad\quad | \quad \mathsf{authPowerUp\text{-}T}(\mathsf{st}_T)$
$\quad\quad | \quad\quad | \quad\quad \llcorner \ \textbf{return } (\perp, \mathsf{true})$
$\quad\quad | \quad\quad | \quad \textbf{else}$
$\quad\quad | \quad\quad | \quad\quad \llcorner \ \textbf{return } (\perp, \mathsf{false})$
$\quad \mathsf{st}_T.m \leftarrow 3, \mathsf{st}_T.\mathsf{pinRetries} \leftarrow \mathsf{pinRetriesMax}$
$\quad \textbf{for all } \mathsf{puvProtocol}' \in \mathsf{st}_T.\mathsf{puvProtocolList} \textbf{ do}$
$\quad\quad \llcorner \ \mathsf{st}_T.\mathsf{puvProtocol}'.\mathsf{resetPuvToken}()$
$\quad \pi_T^i.\mathsf{bs} \leftarrow \pi_T^i.\mathsf{puvProtocol}.pt$
$\quad \textbf{if } (\mathsf{active}_{\mathsf{key}} = \mathsf{true} \vee \mathsf{count}_{\mathsf{execute}} > \mathsf{hy}) \wedge$
$\quad (\mathsf{st}_T.\mathsf{puvProtocol}.pk, c) \neq \mathcal{L}_{\mathsf{setupKeys}}[T] \textbf{ then}$
$\quad\quad \llcorner \ c_{pt} \xleftarrow{\$} \mathsf{st}_T.\mathsf{puvProtocol}.\mathsf{encrypt}(K, \pi_T^i.\mathsf{bs})$
$\quad \textbf{else}$
$\quad\quad | \quad \tilde{pt} \xleftarrow{\$} \{0, 1\}^{len(\pi_T^i.\mathsf{bs})}$
$\quad\quad | \quad c_{pt} \xleftarrow{\$} \mathsf{st}_T.\mathsf{puvProtocol}.\mathsf{encrypt}(K, \tilde{pt})$
$\quad\quad \llcorner \ \mathcal{L}_{\mathsf{ct}}[\_, \pi_T^i] \leftarrow (c', c'_{ph}, \pi_T^i.\mathsf{bs})$
$\quad \pi_T^i.\mathsf{st}_{\mathsf{exe}} \leftarrow \mathsf{bindDone}$
$\quad \pi_T^i.\mathsf{canValidate} \leftarrow \mathsf{true}$
$\quad \pi_T^i.\mathsf{sid} \leftarrow \pi_T^i.\mathsf{sid} \ || \ \mathsf{puvProtocol} \ || \ c \ || \ c_{ph} \ || \ c_{pt} \ || \ \mathsf{false}$
$\quad \textbf{return } (c_{pt}, \mathsf{false})$

**Fig. 56.** Hybrid game 5.$i$, Execute oracle and modified CTAP 2.1+ functions obtainPinUvAuthToken-C-start, obtainPinUvAuthToken-C-end and obtainPinUvAuthToken-T from hybrid game 5.$i$.

$\mathcal{B}_6^i(\mathcal{A}, \mathcal{O}_{\mathsf{LR}}, \mathcal{O}_{\mathsf{RAND}}, \mathcal{O}_{\mathsf{LPC}})$:

  $\mathcal{L}_{\mathsf{authC}}, \mathcal{L}_{\mathsf{authT}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{valid}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{corrupt}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{sCDH}}, \mathcal{L}_{\mathcal{H}_2} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{TokenSetup}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{pk}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{hash}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{pin}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{setupKeys}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{ct}} \leftarrow \emptyset$
  $\mathsf{hy} \leftarrow i, \mathsf{count}_{\mathsf{executePk}} \leftarrow 0, \mathcal{L}_{\mathsf{execute}} \leftarrow \emptyset$
  $\pi_C^* \leftarrow \perp, \pi_T^* \leftarrow \perp$
  $\mathsf{win\text{-}SUF\text{-}t} \leftarrow 0$
  $() \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(1^\lambda)$
  $b' \leftarrow \perp$
  **if** $\mathsf{win\text{-}SUF\text{-}t} = 1$ **then**
    **return** $b' \leftarrow 0$
  **else**
    **return** $b' \leftarrow 1$
  **return** $b'$

Execute $(T, i, C, j)$:

  $\mathsf{pin}_U \leftarrow \mathcal{L}_{\mathsf{valid}}[U]$
  **if** $\mathsf{st}_T = \perp$ **or** $\pi_T^i \neq \perp$ **or** $\pi_C^j \neq \perp$ **or** $\mathsf{pin}_U = \perp$ **then**
    **return** $\perp$
  $\pi_T^i \leftarrow \mathsf{st}_T$
  $\mathsf{trans}, m_C, m_T \leftarrow \perp$
  $\mathcal{L}_{\mathsf{ct}}[\pi_C^j, \pi_T^i] \leftarrow (\perp, \perp, \perp)$
  $\mathsf{count}_{\mathsf{execute}} \leftarrow \mathsf{count}_{\mathsf{execute}} + 1$
  $\pi_C^* \leftarrow \pi_C^j, \pi_T^* \leftarrow \pi_T^i$
  **while** $\pi_C^j.\mathsf{st}_{\mathsf{exe}} \neq \mathsf{bindStart}$ **do**
    $m_T \xleftarrow{\$} \mathsf{Bind\text{-}T}(\pi_T^i, m_C)$
    $m_C \xleftarrow{\$} \mathsf{Bind\text{-}C}(\pi_C^j, U, m_T)$
    $\mathsf{trans} \leftarrow \mathsf{trans} \,||\, m_T \,||\, m_C$
  **return** $\mathsf{trans}$

obtainPinUvAuthToken-C-start$(\pi_C^j, \mathsf{pin})$:

  **if** $\mathsf{count} = \mathsf{hy}$ **then**
    $c_{ph} \xleftarrow{\$} \mathcal{O}_{\mathsf{LR}}(\mathsf{H}_\mathsf{w}(\mathsf{pin}), \mathsf{H}_\mathsf{w}(0000))$
  **else if** $\mathsf{count}_{\mathsf{execute}} \leq \mathsf{hy}$ **then**
    $\mathsf{pinHash} \leftarrow \mathsf{H}_\mathsf{w}(0000)$
    $c_{ph} \xleftarrow{\$} \pi_C^j.\mathsf{puvProtocol.encrypt}(\pi_C^j.\mathsf{K}, \mathsf{pinHash})$
  **else**
    $\mathsf{pinHash} \leftarrow \mathsf{H}_\mathsf{w}(\mathsf{pin})$
    $c_{ph} \xleftarrow{\$} \pi_C^j.\mathsf{puvProtocol.encrypt}(\pi_C^j.\mathsf{K}, \mathsf{pinHash})$
  $pk_{\mathsf{client}} \leftarrow \pi_C^*.\mathsf{puvProtocol}.pk$
  $pk_{\mathsf{token}} \leftarrow \pi_T^*.\mathsf{puvProtocol}.pk$
  $\mathcal{L}_{\mathsf{executePk}}[pk_{\mathsf{client}}, pk_{\mathsf{token}}] \leftarrow \mathsf{count}_{\mathsf{execute}}$
  $\pi_C^j.\mathsf{st}_{\mathsf{exe}} \leftarrow \mathsf{bindStart}$
  $\pi_C^j.\mathsf{sid} \leftarrow \pi_C^j.\mathsf{sid} \,||\, c_{ph}$
  $\mathcal{L}_{\mathsf{ct}}[\pi_C^j, \_] \leftarrow (\pi_C^j.\mathsf{puvProtocol}.pk, c_{ph}, \perp)$
  **return** $c_{ph}$

obtainPinUvAuthToken-C-end$(\pi_C^j, c_{pt})$:

  **if** $\mathsf{count}_{\mathsf{execute}} \leq \mathsf{hy}$ **then**
    $(\_, \_, pt) \leftarrow \mathcal{L}_{\mathsf{ct}}[\pi_C^j, \_]$
    $\pi_C^j.\mathsf{bs} \leftarrow pt$
  **else**
    $\pi_C^j.\mathsf{bs} \leftarrow \pi_C^j.\mathsf{puvProtocol.decrypt}(\pi_C^j.\mathsf{K}, c_{pt})$
  $\pi_C^j.\mathsf{st}_{\mathsf{exe}} \leftarrow \mathsf{bindDone}$
  $\pi_C^j.\mathsf{sid} \leftarrow \pi_C^j.\mathsf{sid} \,||\, c_{pt} \,||\, \mathsf{false}$
  **return**

obtainPinUvAuthToken-T$(\pi_T^i, \mathsf{puvProtocol}, c, c_{ph})$:

  **if** $\mathsf{puvProtocol} \notin \mathsf{st}_T.\mathsf{puvProtocolList} \vee \mathsf{st}_T.\mathsf{pinRetries} = 0$
  **then**
    **return** $(\perp, \mathsf{false})$
  $K \leftarrow \mathsf{st}_T.\mathsf{puvProtocol.decapsulate}(c)$
  **if** $K = \perp$ **then**
    **return** $(\perp, \mathsf{false})$
  $\mathsf{st}_T.\mathsf{pinRetries} \leftarrow \mathsf{pinRetries} - 1$
  $\mathsf{active}_{\mathsf{key}}, \mathsf{active}_{c_{ph}} \leftarrow \mathsf{true}$
  **for all** $(\pi_{C'}^{j'}, \pi_{T'}^{i'}, c', c'_{ph}, \_) \in \mathcal{L}_{\mathsf{ct}}$ s.t. $T' = T \wedge$
  $\pi_{T'}^{i'}.\mathsf{puvProtocol}.pk = \pi_T^i.\mathsf{puvProtocol}.pk$ **do**
    **if** $c' = c$ **then**
      $\mathsf{active}_{\mathsf{key}} \leftarrow \mathsf{false}$
    **if** $c'_{ph} = c_{ph} \wedge c' = c$ **then**
      $\mathsf{active}_{c_{ph}} \leftarrow \mathsf{false}$
      **break**
  $\mathsf{verifyPin} \leftarrow \mathsf{active}_{\mathsf{key}} \vee \mathsf{active}_{c_{ph}}$
  $\mathsf{count}_{\mathsf{execute}} \leftarrow \mathcal{L}_{\mathsf{executePk}}[c, \pi_T^i.\mathsf{puvProtocol}.pk]$
  **if** $\mathsf{verifyPin} = \mathsf{true} \vee \mathsf{count}_{\mathsf{execute}} > \mathsf{hy}$ **then**
    $\mathsf{pinHash}_T \leftarrow \mathcal{L}_{\mathsf{TokenSetup}}[T]$
    **if** $\mathsf{count}_{\mathsf{execute}} = \mathsf{hy}$ **then**
      **if** $\mathcal{O}_{\mathsf{LPC}}(c_{ph}) = \mathsf{true}$ **then**
        $\mathsf{pinHash} \leftarrow \mathsf{pinHash}_T$
      **else**
        $\mathsf{pinHash} \leftarrow \perp$
    **else**
      $\mathsf{pinHash} \leftarrow \mathsf{st}_T.\mathsf{puvProtocol.decrypt}(K, c_{ph})$
    **if** $\mathsf{pinHash} \neq \mathsf{pinHash}_T$ **then**
      $\mathsf{st}_T.\mathsf{puvProtocol.regenerate}()$
      $\mathsf{st}_T.m \leftarrow \mathsf{st}_T.m - 1$
      **if** $\mathsf{st}_T.m = 0$ **then**
        $\mathsf{authPowerUp\text{-}T}(\mathsf{st}_T)$
        **return** $(\perp, \mathsf{true})$
      **else**
        **return** $(\perp, \mathsf{false})$
  $\mathsf{st}_T.m \leftarrow 3, \mathsf{st}_T.\mathsf{pinRetries} \leftarrow \mathsf{pinRetriesMax}$
  **for all** $\mathsf{puvProtocol}' \in \mathsf{st}_T.\mathsf{puvProtocolList}$ **do**
    $\mathsf{st}_T.\mathsf{puvProtocol}'.\mathsf{resetPuvToken}()$
  $\pi_T^i.\mathsf{bs} \leftarrow \pi_T^i.\mathsf{puvProtocol}.pt$
  **if** $(\mathsf{active}_{\mathsf{key}} = \mathsf{true} \vee \mathsf{count}_{\mathsf{execute}} > \mathsf{hy}) \wedge$
  $(\mathsf{st}_T.\mathsf{puvProtocol}.pk, c) \neq \mathcal{L}_{\mathsf{setupKeys}}[T]$ **then**
    $c_{pt} \xleftarrow{\$} \mathsf{st}_T.\mathsf{puvProtocol.encrypt}(K, \pi_T^i.\mathsf{bs})$
  **else if** $\mathsf{count}_{\mathsf{execute}} = \mathsf{hy}$ **then**
    $(pt_0, pt_1, c_{pt}) \xleftarrow{\$} \mathcal{O}_{\mathsf{RAND}}(\mathsf{len}(\pi_T^i.\mathsf{bs}))$
    $\pi_T^i.\mathsf{bs} \leftarrow pt_0$
    $\mathcal{L}_{\mathsf{ct}}[\_, \pi_T^i] \leftarrow (c', c'_{ph}, \pi_T^i.\mathsf{bs})$
  **else**
    $\tilde{pt} \xleftarrow{\$} \{0,1\}^{len(\pi_T^i.\mathsf{bs})}$
    $c_{pt} \xleftarrow{\$} \mathsf{st}_T.\mathsf{puvProtocol.encrypt}(K, \tilde{pt})$
    $\mathcal{L}_{\mathsf{ct}}[\_, \pi_T^i] \leftarrow (c', c'_{ph}, \pi_T^i.\mathsf{bs})$
  $\pi_T^i.\mathsf{st}_{\mathsf{exe}} \leftarrow \mathsf{bindDone}$
  $\pi_T^i.\mathsf{canValidate} \leftarrow \mathsf{true}$
  $\pi_T^i.\mathsf{sid} \leftarrow \pi_T^i.\mathsf{sid} \,||\, \mathsf{puvProtocol} \,||\, c \,||\, c_{ph} \,||\, c_{pt} \,||\, \mathsf{false}$
  **return** $(c_{pt}, \mathsf{false})$

**Fig. 57.** Adversary $\mathcal{B}_6^i$'s simulation of game 5.$i$, Execute oracle and modified CTAP 2.1+ functions obtainPinUvAuthToken-C-start, obtainPinUvAuthToken-C-end and obtainPinUvAuthToken-T.

$$n_{\text{execute}} \ \mathsf{Adv}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LPC}}(\mathcal{B}_6) \geq \sum_{i=1}^{n_{\text{execute}}} \mathsf{Adv}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LPC}}(\mathcal{B}_6^i) \tag{54}$$

It then follows that:

$$n_{\text{execute}} \ \mathsf{Adv}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LPC}}(\mathcal{B}_6) \geq \ \mid \mathsf{Pr}_{5.0} - \mathsf{Pr}_{5.n_{\text{execute}}} \mid \tag{55}$$

Since Game 5.0 corresponds to Game 5 and Game $5.n_{\text{execute}}$ corresponds to Game 6, we have:

$$n_{\text{execute}} \ \mathsf{Adv}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LPC}}(\mathcal{B}_6) \geq \ \mid \mathsf{Pr}_5 - \mathsf{Pr}_6 \mid \tag{56}$$

And, since $n_{\text{execute}}$ is the maximum amount of valid Execute queries made by $\mathcal{A}$, we have that $n_{\text{execute}} \leq q_{\mathsf{E}}$. Therefore:

$$q_{\mathsf{E}} \ \mathsf{Adv}_{\mathsf{SKE}}^{\mathsf{IND\text{-}1\$PA\text{-}LPC}}(\mathcal{B}_6) \geq \ \mid \mathsf{Pr}_5 - \mathsf{Pr}_6 \mid \tag{57}$$

**Game 7**

In this game, no user pins are sampled until the pin is necessary to respond to some interaction with $\mathcal{A}$. This happens *only* when $\mathcal{A}$ corrupts a user $U$ to obtain its pin, or when it provides a pinHash of its own when actively attacking token $T$ during the Bind session by calling Send-Bind-T, where the pinHash must be verified.



**Fig. 58.** Game 7, with modified NewU and CorruptUser oracles.

**Advantage measure** The modifications in Game 7 affect *only* when a pin is sampled from a distribution $\mathscr{D}$ over $\mathcal{P}$, and in such a way that it is identical for an adversary to play Game 6 and Game 7.

In fact, since each sampling of a pin is independent from any other sampling of any other pin, it is irrelevant when that sampling occurs, as long as it occurs before the adversary interacts with said pin. Thus, we can sample the pin *exactly* when one of these events occurs, and the game proceeds identically to Game 6.

```
obtainPinUvAuthToken-T(π_T^i, puvProtocol, c, c_ph):
    if puvProtocol ∉ st_T.puvProtocolList ∨ st_T.pinRetries = 0 then
        └ return (⊥, false)
    K ← st_T.puvProtocol.decapsulate(c)
    if K = ⊥ then
        └ return (⊥, false)
    st_T.pinRetries ← pinRetries − 1
    active_key, active_{c_ph} ← true
    for all (π_{C'}^{j'}, π_{T'}^{i'}, c', c_ph', _) ∈ L_ct s.t. T' = T ∧ π_{T'}^{i'}.puvProtocol.pk = π_T^i.puvProtocol.pk
    do
        │ if c' = c then
        │     └ active_key ← false
        │ if c_ph' = c_ph ∧ c' = c then
        │     active_{c_ph} ← false
        │     └ break
    verifyPin ← active_key ∨ active_{c_ph}
    if verifyPin = true then
        │ pinHash ← st_T.puvProtocol.decrypt(K, c_ph)
        │ if L_sampled[st_T.user] = false then
        │     │ pin ←$ P
        │     │ L_valid[st_T.user] ← pin
        │     └ L_sampled[st_T.user] ← true
        │ pinHash_T ← H_w(L_valid[st_T.user])
        │ if pinHash ≠ pinHash_T then
        │     │ st_T.puvProtocol.regenerate()
        │     │ st_T.m ← st_T.m − 1
        │     │ if st_T.m = 0 then
        │     │     │ authPowerUp-T(st_T)
        │     │     └ return (⊥, true)
        │     │ else
        │     └     └ return (⊥, false)
    st_T.m ← 3, st_T.pinRetries ← pinRetriesMax
    for all puvProtocol' ∈ st_T.puvProtocolList do
        └ st_T.puvProtocol'.resetPuvToken()
    π_T^i.bs ← π_T^i.puvProtocol.pt
    if active_key = true ∧ (st_T.puvProtocol.pk, c) ≠ L_setupKeys[T] then
        │ c_pt ←$ st_T.puvProtocol.encrypt(K, π_T^i.bs)
    else
        │ p̃t ←$ {0,1}^{len(π_T^i.bs)}
        │ c_pt ←$ st_T.puvProtocol.encrypt(K, p̃t)
        │ L_ct[_, π_T^i] ← (c', c_ph', π_T^i.bs)
    π_T^i.st_exe ← bindDone
    π_T^i.canValidate ← true
    π_T^i.sid ← π_T^i.sid || puvProtocol || c || c_ph || c_pt || false
    return (c_pt, false)
```

**Fig. 59.** Modified CTAP 2.1+ function obtainPinUvAuthToken-T - Game 7.

Therefore,

$$\mathsf{Pr}_6 = \mathsf{Pr}_7 \tag{58}$$

**Game 8**

In this game, all of $\mathcal{A}'s$ active attempts at guessing the pin for a given token are rejected, unless $\mathcal{A}$ has corrupted the user pin associated with the target token.

Recall that at this point, $\mathcal{A}$ has no information whatsoever about any pin for any token whose user it has not already corrupted. No pin was ever sampled until this point, and all ciphertexts sent by any client session contain constant values.

obtainPinUvAuthToken-T($\pi_T^i$, puvProtocol, $c$, $c_{ph}$):
    **if** puvProtocol $\notin$ st$_T$.puvProtocolList $\vee$ st$_T$.pinRetries $= 0$
    **then**
        **return** ($\bot$, false)
    $K \leftarrow$ st$_T$.puvProtocol.decapsulate($c$)
    **if** $K = \bot$ **then**
        **return** ($\bot$, false)
    st$_T$.pinRetries $\leftarrow$ pinRetries $- 1$
    active$_{key}$, active$_{c_{ph}} \leftarrow$ true
    **for all** $(\pi_{C'}^{j'}, \pi_{T'}^{i'}, c', c_{ph}', \_) \in \mathcal{L}_{ct}$ s.t. $T' = T \wedge$
    $\pi_{T'}^{i'}$.puvProtocol.$pk = \pi_T^i$.puvProtocol.$pk$ **do**
        **if** $c' = c$ **then**
            active$_{key} \leftarrow$ false
        **if** $c_{ph}' = c_{ph} \wedge c' = c$ **then**
            active$_{c_{ph}} \leftarrow$ false
            break
    verifyPin $\leftarrow$ active$_{key} \vee$ active$_{c_{ph}}$
    **if** verifyPin $=$ true **then**
        pinHash $\leftarrow$ st$_T$.puvProtocol.decrypt($K$, $c_{ph}$)
        ▷ *NOTE: The value of $\mathcal{L}_{sampled}[st_T.user]$ will always be*
        *true if the User has been corrupted. Therefore, it is kept*
        *only for simplifying the hybrid game's code.*
        **if** $\mathcal{L}_{corrupt}[$st$_T$.user$] =$ true $\wedge \mathcal{L}_{sampled}[$st$_T$.user$] =$ false
        **then**
            pin $\xleftarrow{\$} \mathcal{P}$
            $\mathcal{L}_{valid}[$st$_T$.user$] \leftarrow$ pin
            $\mathcal{L}_{sampled}[$st$_T$.user$] \leftarrow$ true
        pinHash$_T \leftarrow H_w(\mathcal{L}_{valid}[$st$_T$.user$])$
        **if** $\mathcal{L}_{corrupt}[$st$_T$.user$] =$ false $\vee$ pinHash $\neq$ pinHash$_T$ **then**
            st$_T$.puvProtocol.regenerate()
            st$_T$.m $\leftarrow$ st$_T$.m $- 1$
            **if** st$_T$.m $= 0$ **then**
                authPowerUp-T(st$_T$)
                **return** ($\bot$, true)
            **else**
                **return** ($\bot$, false)

    st$_T$.m $\leftarrow 3$, st$_T$.pinRetries $\leftarrow$ pinRetriesMax
    **for all** puvProtocol' $\in$ st$_T$.puvProtocolList **do**
        st$_T$.puvProtocol'.resetPuvToken()
    $\pi_T^i$.bs $\leftarrow \pi_T^i$.puvProtocol.$pt$
    **if** active$_{key} =$ true $\wedge$ (st$_T$.puvProtocol.$pk$, $c$) $\neq \mathcal{L}_{setupKeys}[T]$
    **then**
        $c_{pt} \xleftarrow{\$}$ st$_T$.puvProtocol.encrypt($K$, $\pi_T^i$.bs)
    **else**
        $\tilde{pt} \xleftarrow{\$} \{0,1\}^{len(\pi_T^i.bs)}$
        $c_{pt} \xleftarrow{\$}$ st$_T$.puvProtocol.encrypt($K$, $\tilde{pt}$)
        $\mathcal{L}_{ct}[\_, \pi_T^i] \leftarrow (c', c_{ph}', \pi_T^i$.bs)
    $\pi_T^i$.st$_{exe} \leftarrow$ bindDone
    $\pi_T^i$.canValidate $\leftarrow$ true
    $\pi_T^i$.sid $\leftarrow \pi_T^i$.sid $||$ puvProtocol $|| c || c_{ph} || c_{pt} ||$ false
    **return** ($c_{pt}$, false)

**Fig. 60.** Modified CTAP 2.1+ obtainPinUvAuthToken-T function - Game 8.

**Hybrid Argument** We bound the advantage of $\mathcal{A}$ against Game 8 by considering $q_{\mathsf{Send}}^{\mathsf{act}}$ hybrid games, where $q_{\mathsf{Send}}^{\mathsf{act}}$ represents the total amount of active queries made by $\mathcal{A}$ to Send-Bind-T (where the pin must be verified). On game 7.$i$, in the first $i^{th}$ active queries to Send-Bind-T, every attempt at guessing the pin is rejected. On the $i^{th}$ attempt, we define a bad event that causes the experiment to abort and $\mathcal{A}$ to lose if $\mathcal{A}$ guesses the correct pin but the challenger still rejects.

**Advantage measure** Recall that during any game 7.$i$, no pin was ever sampled for any token before attempt $i$ (except for when $\mathcal{A}$ corrupts a user, which makes it impossible to cause bad). Therefore, it is always true that:

$$\mathsf{Pr}_{7.i}[\mathsf{bad}] = \frac{1}{2^{h_\mathscr{D}}}, \forall i \in [1, q_{\mathsf{Send}}^{\mathsf{act}}] \tag{59}$$

Since games 7.$i$ and 7.$(i-1)$ are identical until bad, through the difference lemma, we have:

$$\mathsf{Pr}_{7.i}[\mathsf{bad}] \geq |\ \mathsf{Pr}_{7.(i-1)} - \mathsf{Pr}_{7.i}\ |, \forall i \in [1, q_{\mathsf{Send}}^{\mathsf{act}}] \tag{60}$$

$$\begin{array}{l}
\text{Game7}.i(\mathcal{A}): \\
\quad \mathcal{L}_{\mathsf{authC}}, \mathcal{L}_{\mathsf{authT}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{valid}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{corrupt}} \leftarrow \emptyset \\
\quad \mathcal{L}_{\mathsf{sCDH}}, \mathcal{L}_{\mathcal{H}_2} \leftarrow \emptyset \\
\quad \mathcal{L}_{\mathsf{TokenSetup}} \leftarrow \emptyset \\
\quad \mathcal{L}_{\mathsf{pk}} \leftarrow \emptyset \\
\quad \mathcal{L}_{\mathsf{hash}} \leftarrow \emptyset \\
\quad \mathcal{L}_{\mathsf{pin}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{setupKeys}} \leftarrow \emptyset \\
\quad \mathcal{L}_{\mathsf{ct}} \leftarrow \emptyset \\
\quad \mathcal{L}_{\mathsf{sampled}} \leftarrow \emptyset \\
\quad {\color{green} \mathsf{hy} \leftarrow i, \mathsf{count}_{\mathsf{ActSendT}} \leftarrow 0} \\
\quad {\color{red} \mathsf{bad} \leftarrow \mathsf{false}} \\
\quad \mathsf{win\text{-}SUF\text{-}t} \leftarrow 0 \\
\quad () \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(1^\lambda) \\
\quad \mathbf{return}\ \mathsf{win\text{-}SUF\text{-}t}
\end{array}$$

**Fig. 61.** Hybrid game 7.$i$.

$\mathsf{obtainPinUvAuthToken\text{-}T}(\pi_T^i, \mathsf{puvProtocol}, c, c_{ph})$:

$\quad$ **if** $\mathsf{puvProtocol} \notin \mathsf{st}_T.\mathsf{puvProtocolList} \vee \mathsf{st}_T.\mathsf{pinRetries} = 0$
$\quad$ **then**
$\quad\quad \lfloor\ \mathbf{return}\ (\perp, \mathsf{false})$
$\quad K \leftarrow \mathsf{st}_T.\mathsf{puvProtocol}.\mathsf{decapsulate}(c)$
$\quad$ **if** $K = \perp$ **then**
$\quad\quad \lfloor\ \mathbf{return}\ (\perp, \mathsf{false})$
$\quad \mathsf{st}_T.\mathsf{pinRetries} \leftarrow \mathsf{pinRetries} - 1$
$\quad \mathsf{active}_{\mathsf{key}}, \mathsf{active}_{c_{ph}} \leftarrow \mathsf{true}$
$\quad$ **for all** $(\pi_{C'}^{j'}, \pi_{T'}^{i'}, c', c'_{ph}, \_) \in \mathcal{L}_{\mathsf{ct}}$ s.t. $T' = T \wedge$
$\quad \pi_{T'}^{i'}.\mathsf{puvProtocol}.pk = \pi_T^i.\mathsf{puvProtocol}.pk$ **do**
$\quad\quad$ **if** $c' = c$ **then**
$\quad\quad\quad \lfloor\ \mathsf{active}_{\mathsf{key}} \leftarrow \mathsf{false}$
$\quad\quad$ **if** $c'_{ph} = c_{ph} \wedge c' = c$ **then**
$\quad\quad\quad\ \mathsf{active}_{c_{ph}} \leftarrow \mathsf{false}$
$\quad\quad\quad \lfloor\ \mathbf{break}$
$\quad \mathsf{verifyPin} \leftarrow \mathsf{active}_{\mathsf{key}} \vee \mathsf{active}_{c_{ph}}$
$\quad$ **if** $\mathsf{verifyPin} = \mathsf{true}$ **then**
$\quad\quad {\color{green} \mathsf{count}_{\mathsf{ActSendT}} \leftarrow \mathsf{count}_{\mathsf{ActSendT}} + 1}$
$\quad\quad \mathsf{pinHash} \leftarrow \mathsf{st}_T.\mathsf{puvProtocol}.\mathsf{decrypt}(K, c_{ph})$
$\quad\quad$ **if** $({\color{orange} \mathcal{L}_{\mathsf{corrupt}}[\mathsf{st}_T.\mathsf{user}] = \mathsf{true}} \vee {\color{green} \mathsf{count}_{\mathsf{ActSendT}} \geq \mathsf{hy}}) \wedge$
$\quad\quad \mathcal{L}_{\mathsf{sampled}}[\mathsf{st}_T.\mathsf{user}] = \mathsf{false}$ **then**
$\quad\quad\quad \mathsf{pin} \xleftarrow{\$} \mathcal{P}$
$\quad\quad\quad \mathcal{L}_{\mathsf{valid}}[\mathsf{st}_T.\mathsf{user}] \leftarrow \mathsf{pin}$
$\quad\quad\quad \lfloor\ \mathcal{L}_{\mathsf{sampled}}[\mathsf{st}_T.\mathsf{user}] \leftarrow \mathsf{true}$
$\quad\quad \mathsf{pinHash}_T \leftarrow \mathsf{H}_w(\mathcal{L}_{\mathsf{valid}}[\mathsf{st}_T.\mathsf{user}])$
$\quad\quad$ **if** $({\color{orange} \mathcal{L}_{\mathsf{corrupt}}[\mathsf{st}_T.\mathsf{user}] = \mathsf{false}} \wedge {\color{green} \mathsf{count}_{\mathsf{ActSendT}} \leq \mathsf{hy}}) \vee$
$\quad\quad \mathsf{pinHash} \neq \mathsf{pinHash}_T$ **then**
$\quad\quad\quad$ **if** ${\color{green} \mathsf{count}_{\mathsf{ActSendT}} = \mathsf{hy}}$ **then**
$\quad\quad\quad\quad$ **if** ${\color{red} \mathsf{pinHash} = \mathsf{pinHash}_T}$ **then**
$\quad\quad\quad\quad\quad {\color{red} \mathsf{bad} \leftarrow \mathsf{true}}$
$\quad\quad\quad\quad \lfloor\ \lfloor\ {\color{red} \mathbf{abort}}$
$\quad\quad\quad \mathsf{st}_T.\mathsf{puvProtocol}.\mathsf{regenerate}()$
$\quad\quad\quad \mathsf{st}_T.m \leftarrow \mathsf{st}_T.m - 1$
$\quad\quad\quad$ **if** $\mathsf{st}_T.m = 0$ **then**
$\quad\quad\quad\quad \mathsf{authPowerUp\text{-}T}(\mathsf{st}_T)$
$\quad\quad\quad\quad \lfloor\ \mathbf{return}\ (\perp, \mathsf{true})$
$\quad\quad\quad$ **else**
$\quad\quad\quad\quad \lfloor\ \mathbf{return}\ (\perp, \mathsf{false})$

$\mathsf{st}_T.m \leftarrow 3, \mathsf{st}_T.\mathsf{pinRetries} \leftarrow \mathsf{pinRetriesMax}$
**for all** $\mathsf{puvProtocol}' \in \mathsf{st}_T.\mathsf{puvProtocolList}$ **do**
$\quad \lfloor\ \mathsf{st}_T.\mathsf{puvProtocol}'.\mathsf{resetPuvToken}()$
$\pi_T^i.\mathsf{bs} \leftarrow \pi_T^i.\mathsf{puvProtocol}.pt$
**if** $\mathsf{active}_{\mathsf{key}} = \mathsf{true} \wedge (\mathsf{st}_T.\mathsf{puvProtocol}.pk, c) \neq \mathcal{L}_{\mathsf{setupKeys}}[T]$
**then**
$\quad \lfloor\ c_{pt} \xleftarrow{\$} \mathsf{st}_T.\mathsf{puvProtocol}.\mathsf{encrypt}(K, \pi_T^i.\mathsf{bs})$
**else**
$\quad \tilde{pt} \xleftarrow{\$} \{0,1\}^{len(\pi_T^i.\mathsf{bs})}$
$\quad c_{pt} \xleftarrow{\$} \mathsf{st}_T.\mathsf{puvProtocol}.\mathsf{encrypt}(K, \tilde{pt})$
$\quad \lfloor\ \mathcal{L}_{\mathsf{ct}}[\_, \pi_T^i] \leftarrow (c', c'_{ph}, \pi_T^i.\mathsf{bs})$
$\pi_T^i.\mathsf{st}_{\mathsf{exe}} \leftarrow \mathsf{bindDone}$
$\pi_T^i.\mathsf{canValidate} \leftarrow \mathsf{true}$
$\pi_T^i.\mathsf{sid} \leftarrow \pi_T^i.\mathsf{sid} \parallel \mathsf{puvProtocol} \parallel c \parallel c_{ph} \parallel c_{pt} \parallel \mathsf{false}$
**return** $(c_{pt}, \mathsf{false})$

**Fig. 62.** Modified CTAP 2.1+ function obtainPinUvAuthToken-T - Game 7.$i$.

$$\sum_{i=1}^{q_{\mathsf{Send}}^{\mathsf{act}}} \Pr_{7.i}[\mathsf{bad}] \geq \sum_{i=1}^{q_{\mathsf{Send}}^{\mathsf{act}}} |\Pr_{7.(i-1)} - \Pr_{7.i}| \tag{61}$$

$$\sum_{i=1}^{q_{\mathsf{Send}}^{\mathsf{act}}} \Pr_{7.i}[\mathsf{bad}] \geq |\Pr_0 - \Pr_{7.q_{\mathsf{Send}}^{\mathsf{act}}}| \tag{62}$$

Since $\mathsf{bad}$ has the same probability of $1/2^{h_{\mathscr{D}}}$ for every $i \in [1, q_{\mathsf{Send}}^{\mathsf{act}}]$, we can write:

$$q_{\mathsf{Send}}^{\mathsf{act}}/2^{h_{\mathscr{D}}} \geq |\Pr_0 - \Pr_{7.q_{\mathsf{Send}}^{\mathsf{act}}}| \tag{63}$$

Since Game 7.0 corresponds to Game 7 and Game $7.q_{\mathsf{Send}}^{\mathsf{act}}$ corresponds to Game 8, we have:

$$q_{\mathsf{Send}}^{\mathsf{act}}/2^{h_{\mathscr{D}}} \geq |\Pr_7 - \Pr_8| \tag{64}$$

**Game 9**

In this game, $\mathcal{A}$ loses by causing a $\mathsf{bad}$ event if and only if a collision between two pinTokens that are sent (encrypted) to $\mathcal{A}$ occurs. More specifically, we are only considering pinTokens that i. are generated by an actively attacked token session $\pi_T^i$ that gives the pinToken directly to $\mathcal{A}$, which happens only in Send-Bind-T queries, or ii. are generated by an honest execution of Bind through an Execute query, in which case the pinToken being considered is the random value $\tilde{pt}$ that is sent in ciphertext $\tilde{c}_{pt}$ and not the pinToken $pt$ set as the binding state (recall that this $\tilde{c}_{pt}$ will be part of the session identifier).

In either case, each query to Execute or Send-Bind-T will always, at most, result in one pinToken that is sent (encrypted) to $\mathcal{A}$.

---

Game9($\mathcal{A}$):
    $\mathcal{L}_{\mathsf{authC}}, \mathcal{L}_{\mathsf{authT}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{valid}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{corrupt}} \leftarrow \emptyset$
    $\mathcal{L}_{\mathsf{sCDH}}, \mathcal{L}_{\mathcal{H}_2} \leftarrow \emptyset$
    $\mathcal{L}_{\mathsf{TokenSetup}} \leftarrow \emptyset$
    $\mathcal{L}_{\mathsf{pk}} \leftarrow \emptyset$
    $\mathcal{L}_{\mathsf{hash}} \leftarrow \emptyset$
    $\mathcal{L}_{\mathsf{pin}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{setupKeys}} \leftarrow \emptyset$
    $\mathcal{L}_{\mathsf{ct}} \leftarrow \emptyset$
    $\mathcal{L}_{\mathsf{sampled}} \leftarrow \emptyset$
    $\mathcal{L}_{\mathsf{pt}} \leftarrow \emptyset$
    $\mathsf{bad} \leftarrow \mathsf{false}$
    $\mathsf{win\text{-}SUF\text{-}t} \leftarrow 0$
    $() \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(1^\lambda)$
    **return** $\mathsf{win\text{-}SUF\text{-}t}$

---

**Fig. 63.** Game 9. The flag $\mathsf{bad}$ in red is only used in this game, to signal when a pinToken collision happens.

```
obtainPinUvAuthToken-T(π_T^i, puvProtocol, c, c_ph):          st_T.m ← 3, st_T.pinRetries ← pinRetriesMax
    if puvProtocol ∉ st_T.puvProtocolList ∨ st_T.pinRetries = 0      for all puvProtocol' ∈ st_T.puvProtocolList do
    then                                                                    st_T.puvProtocol'.resetPuvToken()
        return (⊥, false)                                            π_T^i.bs ← π_T^i.puvProtocol.pt
    K ← st_T.puvProtocol.decapsulate(c)                          if active_key = true ∧ (st_T.puvProtocol.pk, c) ≠ L_setupKeys[T]
    if K =⊥ then                                                 then
        return (⊥, false)                                            if π_T^i.bs ∈ L_pt then
    st_T.pinRetries ← pinRetries − 1                                     bad ← true
    (c', c'_ph, -, -) ← L_ct[-, π_T^i]                                   abort
    active_key ← c' ≠ c                                              L_pt ← L_pt ∪ {π_T^i.bs}
    active_cph ← c'_ph ≠ c_ph                                        c_pt ←$ st_T.puvProtocol.encrypt(K, π_T^i.bs)
    for all (π_C'^j', π_T'^i', c', c'_ph, -, -) ∈ L_ct s.t. T' = T do   else
        if c'_ph = c_ph ∧ c' = c then                                   pt̃ ←$ {0,1}^len(π_T^i.bs)
            active_cph ← false                                          if pt̃ ∈ L_pt then
            active_key ← false                                              bad ← true
    verifyPin ← active_key ∨ active_cph                                     abort
    if verifyPin = true then                                            L_pt ← L_pt ∪ {pt̃}
        pinHash ← st_T.puvProtocol.decrypt(K, c_ph)                     c_pt ←$ st_T.puvProtocol.encrypt(K, pt̃)
        if L_corrupt[st_T.user] = true ∧ L_sampled[st_T.user] = false      L_ct[-, π_T^i] ← (c', c'_ph, c_pt, π_T^i.bs)
        then                                                         π_T^i.st_exe ← bindDone
            pin ←$ P                                                 π_T^i.canValidate ← true
            L_valid[st_T.user] ← pin                                 π_T^i.sid ← π_T^i.sid || puvProtocol || c || c_ph || c_pt || false
            L_sampled[st_T.user] ← true                             return (c_pt, false)
        pinHash_T ← H_w(L_valid[st_T.user])
        if L_corrupt[st_T.user] = false ∨ pinHash ≠ pinHash_T then
            st_T.puvProtocol.regenerate()
            st_T.m ← st_T.m − 1
            if st_T.m = 0 then
                authPowerUp-T(st_T)
                return (⊥, true)
            else
                return (⊥, false)
```

**Fig. 64.** Modified CTAP 2.1+ function obtainPinUvAuthToken-T - Game 9.

**Advantage measure** Since each query to Execute or Send-Bind-T creates and sends to $\mathcal{A}$, at most, one pinToken, the maximum amount of such pinTokens being created in the experiment is $q_E + q_{Send}$.

Since the pinToken is a random byte string of length $2\lambda$ (for the PIN/UV Auth Protocol being analyzed), the maximum number of pinTokens that can possibly exist is $2^{2\lambda}$.

Considering the probability of a collision between two pinTokens as $1/2^{2\lambda}$, and $(q_E + q_{Send})^2/2$ as an upper bound for the amount of pinToken pairs, we have:

$$\Pr_9[\text{bad}] \leq \frac{(q_E + q_{Send})^2}{2^{2\lambda+1}} \tag{65}$$

Since games 8 and 9 are identical until bad, by the difference lemma, we have:

$$\Pr_9[\text{bad}] \geq |\Pr_8 - \Pr_9| \tag{66}$$

And, therefore:

$$\frac{(q_E + q_{Send})^2}{2^{2\lambda+1}} \geq |\Pr_8 - \Pr_9| \tag{67}$$

## Game 10

In this game, $\mathcal{A}$ loses if, at any point in the game, it causes a bad event, which happens when it queries random oracle $\mathcal{H}_3$ with *any* pinToken generated throughout the experiment and set as the binding state of some token or client session, but for which the adversary has no information.

We maintain two new structures $\mathcal{L}_{\mathsf{bs}}$ and $\mathcal{L}_{\mathcal{H}_3}$ to keep track of all binding states about which $\mathcal{A}$ has no information and of all of $\mathcal{A}$'s queries to $\mathcal{H}_3$, respectively.

---

Game10($\mathcal{A}$):
  $\mathcal{L}_{\mathsf{authC}}, \mathcal{L}_{\mathsf{authT}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{valid}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{corrupt}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{sCDH}}, \mathcal{L}_{\mathcal{H}_2} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{TokenSetup}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{pk}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{hash}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{pin}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{setupKeys}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{ct}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{sampled}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{pt}} \leftarrow \emptyset$
  <span style="color:orange">$\mathcal{L}_{\mathsf{bs}}, \mathcal{L}_{\mathcal{H}_3} \leftarrow \emptyset$</span>
  <span style="color:red">bad $\leftarrow$ false</span>
  win-SUF-t $\leftarrow 0$
  $() \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(1^\lambda)$
  return win-SUF-t

Adv$\mathcal{H}_3$ ($u_1$):
  $v \leftarrow \mathcal{H}_3(u_1)$
  if <span style="color:orange">$u_1 \in \mathcal{L}_{\mathsf{bs}}$</span> then
    <span style="color:red">bad $\leftarrow$ true</span>
    <span style="color:orange">abort</span>
  <span style="color:orange">$\mathcal{L}_{\mathcal{H}_3}[u_1] \leftarrow v$</span>
  return $v$

obtainPinUvAuthToken-T($\pi_T^i$, puvProtocol, $c, c_{ph}$):
  if puvProtocol $\notin$ st$_T$.puvProtocolList $\vee$ st$_T$.pinRetries $= 0$
  then
    return $(\bot, \mathsf{false})$
  $K \leftarrow$ st$_T$.puvProtocol.decapsulate$(c)$
  if $K = \bot$ then
    return $(\bot, \mathsf{false})$
  st$_T$.pinRetries $\leftarrow$ pinRetries $- 1$
  $(c', c'_{ph}, \text{-}, \text{-}) \leftarrow \mathcal{L}_{\mathsf{ct}}[\text{-}, \pi_T^i]$
  active$_{\mathsf{key}} \leftarrow c' \neq c$
  active$_{c_{ph}} \leftarrow c'_{ph} \neq c_{ph}$
  for all $(\pi_{C''}^{j'}, \pi_{T'}^{i'}, c', c'_{ph}, \text{-}, \text{-}) \in \mathcal{L}_{\mathsf{ct}}$ s.t. $T' = T$ do
    if $c'_{ph} = c_{ph} \wedge c' = c$ then
      active$_{c_{ph}} \leftarrow$ false
      active$_{\mathsf{key}} \leftarrow$ false
  verifyPin $\leftarrow$ active$_{\mathsf{key}} \vee$ active$_{c_{ph}}$
  if verifyPin $=$ true then
    pinHash $\leftarrow$ st$_T$.puvProtocol.decrypt$(K, c_{ph})$
    if $\mathcal{L}_{\mathsf{corrupt}}[$st$_T$.user$] =$ true $\wedge \mathcal{L}_{\mathsf{sampled}}[$st$_T$.user$] =$ false
    then
      pin $\xleftarrow{\$} \mathcal{P}$
      $\mathcal{L}_{\mathsf{valid}}[$st$_T$.user$] \leftarrow$ pin
      $\mathcal{L}_{\mathsf{sampled}}[$st$_T$.user$] \leftarrow$ true
    pinHash$_T \leftarrow \mathsf{H}_w(\mathcal{L}_{\mathsf{valid}}[$st$_T$.user$])$
    if $\mathcal{L}_{\mathsf{corrupt}}[$st$_T$.user$] =$ false $\vee$ pinHash $\neq$ pinHash$_T$ then
      st$_T$.puvProtocol.regenerate()
      st$_T.m \leftarrow$ st$_T.m - 1$
      if st$_T.m = 0$ then
        authPowerUp-T(st$_T$)
        return $(\bot, \mathsf{true})$
      else
        return $(\bot, \mathsf{false})$

  st$_T.m \leftarrow 3$, st$_T$.pinRetries $\leftarrow$ pinRetriesMax
  for all puvProtocol' $\in$ st$_T$.puvProtocolList do
    st$_T$.puvProtocol'.resetPuvToken()
  $\pi_T^i$.bs $\leftarrow \pi_T^i$.puvProtocol.$pt$
  if active$_{\mathsf{key}} =$ true $\wedge$ (st$_T$.puvProtocol.$pk, c) \neq \mathcal{L}_{\mathsf{setupKeys}}[T]$
  then
    if $\pi_T^i$.bs $\in \mathcal{L}_{\mathsf{pt}}$ then
      abort
    $\mathcal{L}_{\mathsf{pt}} \leftarrow \mathcal{L}_{\mathsf{pt}} \cup \{\pi_T^i$.bs$\}$
    $c_{pt} \xleftarrow{\$}$ st$_T$.puvProtocol.encrypt$(K, \pi_T^i$.bs$)$
  else
    $\tilde{pt} \xleftarrow{\$} \{0,1\}^{len(\pi_T^i.\mathsf{bs})}$
    if $\tilde{pt} \in \mathcal{L}_{\mathsf{pt}}$ then
      abort
    $\mathcal{L}_{\mathsf{pt}} \leftarrow \mathcal{L}_{\mathsf{pt}} \cup \{\tilde{pt}\}$
    if $\pi_T^i$.bs $\in \mathcal{L}_{\mathcal{H}_3}$ then
      <span style="color:red">bad $\leftarrow$ true</span>
      abort
    <span style="color:orange">$\mathcal{L}_{\mathsf{bs}} \leftarrow \mathcal{L}_{\mathsf{bs}} \cup \{\pi_T^i$.bs$\}$</span>
    $c_{pt} \xleftarrow{\$}$ st$_T$.puvProtocol.encrypt$(K, \tilde{pt})$
    $\mathcal{L}_{\mathsf{ct}}[\text{-}, \pi_T^i] \leftarrow (c', c'_{ph}, c_{pt}, \pi_T^i$.bs$)$
  $\pi_T^i$.st$_{\mathsf{exe}} \leftarrow$ bindDone
  $\pi_T^i$.canValidate $\leftarrow$ true
  $\pi_T^i$.sid $\leftarrow \pi_T^i$.sid $||$ puvProtocol $|| c || c_{ph} || c_{pt} ||$ false
  return $(c_{pt}, \mathsf{false})$

**Fig. 65.** Game 10. The flag bad in red is only used in this game.

---

**Advantage measure** At this point in the game, $\mathcal{A}$ has no information about any binding state that results from Execute queries or Send-Bind-T queries where it behaves passively, because in these circumstances the $c_{pt}$ given to $\mathcal{A}$ no longer contains any information about

$\pi_T^i$.bs. Therefore, we can upper bound the amount of binding states that can cause the bad event as $(q_E + q_{\mathsf{Send}})$.

Let $q_{\mathcal{H}_3}$ be the maximum amount of queries $\mathcal{A}$ performs during game 10. Given that the maximum amount of pinTokens that can possibly exist is $2^{2\lambda}$, we know that:

$$\Pr{}_{10}[\mathsf{bad}] \leq q_{\mathcal{H}_3} \ (q_E + q_{\mathsf{Send}})/2^{2\lambda} \tag{68}$$

Since games 9 and 10 are identical until bad, by the difference lemma, we have:

$$\Pr{}_{10}[\mathsf{bad}] \geq |\ \Pr{}_9 - \Pr{}_{10}\ | \tag{69}$$

And, therefore:

$$q_{\mathcal{H}_3} \ (q_E + q_{\mathsf{Send}})/2^{2\lambda} \geq |\ \Pr{}_9 - \Pr{}_{10}\ | \tag{70}$$

**Game 11**

In this game, when $\mathcal{A}$ delivers a message $M$ and a tag $t$ to a token session $\pi_T^i$ that finished Bind passively ($\pi_T^i$ did not finish Bind through an active Send-Bind-T query) by querying oracle Validate-T, the challenger always rejects, provided $M$ and $t$ were not output by any of $\pi_T^i$'s partner (if it has one), the partner was not compromised and the pin associated with $\pi_T^i$ was not corrupted. In sum, given $M$ and $t$ that would result in $\mathcal{A}$ winning the experiment, the challenger rejects instead, making it impossible for $\mathcal{A}$ to win by forging a message tag and delivering it to any token session. We flag this event by creating a bad event $\mathsf{bad}_{11}$.

```
Game11(A):                                          Token-Win-SUF-t (T, i, M, t, d):
    L_authC, L_authT ← ∅, L_valid ← ∅, L_corrupt ← ∅    if d ≠ accepted then return 1
    L_sCDH, L_H₂ ← ∅
    L_TokenSetup ← ∅                                   if ∃(C₁, j₁), (C₂, j₂) s.t. (C₁, j₁) ≠ (C₂, j₂) and π_{C₁}^{j₁}.st_exe =
    L_pk ← ∅                                            π_{C₂}^{j₂}.st_exe = bindDone
    L_hash ← ∅                                          and π_{C₁}^{j₁}.sid = π_{C₂}^{j₂}.sid then return 1
    L_pin ← ∅, L_setupKeys ← ∅
    L_ct ← ∅                                            if ∃(T₁, i₁), (T₂, i₂) s.t. (T₁, i₁) ≠ (T₂, i₂) and π_{T₁}^{i₁}.st_exe =
    L_sampled ← ∅                                       π_{T₂}^{i₂}.st_exe = bindDone
    L_pt ← ∅                                            and π_{T₁}^{i₁}.sid = π_{T₂}^{i₂}.sid then return 1
    L_bs, L_H₃ ← ∅
    bad₁₁ ← false                                       (C, j) ← tokenBindPartner(T, i)
    win-SUF-t ← 0                                       if (C, j, M, t) ∉ L_authC then
    () ←$ A^O(1^λ)                                          if (C, j) = (⊥, ⊥) or π_C^j.compromised = false then
    return win-SUF-t                                           if π_T^i.pinCorr = false then
                                                                   return 1 ∧ bad₁₁ ≠ true

                                                        return 0
```

**Fig. 66.** Game 11 and modified function Token-Win-SUF-t.

Notice that, after **Game 9**, every token session that finishes Bind has *at most* one partner. Indeed, for two sessions $\pi_C^j$ and $\pi_T^i$ to be partnered, they must have finished binding and have matching sid values ($\pi_C^j.\mathsf{sid} = \pi_T^i.\mathsf{sid}$). The sid, after binding is completed, is mainly a concatenation of the Bind session's client and token ECDH public keys $c$ and $pk$, the pinHash

```
Auth-C (C, j, M):                              Validate-T (T, i, M, t, d):
    if π_C^j = ⊥ or π_C^j.st_exe ≠ bindDone or     if π_T^i = ⊥ or π_T^i.st_exe ≠ bindDone or π_T^i.isValid = false then
    π_C^j.isValid = false then                         return ⊥
        return ⊥                                   (C, j) ← tokenBindPartner(T, i)
    (M, t) ←$ auth-C(π_C^j, M)                     status ← validate-T(π_T^i, M, t, d)
    L_authC ← L_authC ∪ {(C, j, M, t)}             if (C, j, M, t) ∉ L_authC then
    return (M, t)                                      if (C, j) = (⊥, ⊥) or π_C^j.compromised = false then
                                                           if π_T^i.pinCorr = false then
                                                               if status = accepted then
                                                                   bad_11 ← true
                                                                   win-SUF-t ← Token-Win-SUF-t(T, i, M, t, d)
                                                               return rejected
                                                   if status = accepted then
                                                       win-SUF-t ← Token-Win-SUF-t(T, i, M, t, d)
                                                   return status
```

**Fig. 67.** Modified Auth-C and Validate-T oracles - Game 11.

ciphertext $c_{ph}$ and the pinToken ciphertext $c_{pt}$. Since we have eliminated collisions of client ECDH public keys $c$ in Game 3, we know that every client session sid is unique. For token-side sid values on the other hand, it is still possible that two or more token sessions of the same token $T$ had the same $c$ delivered to them by $\mathcal{A}$ and the same token ECDH public key for multiple Bind sessions (since the same token can reuse the same public key), which means avoiding collisions between token public keys is not enought to guarantee a unique sid for each token session.

However, in Game 9, we have also eliminated collisions between every pinToken that is encrypted into a ciphertext $c_{pt}$ and sent to $\mathcal{A}$ in Execute or Send-Bind-T queries, which means that $c_{pt}$ can never repeat for the same $c$ and $pk$, as long as the underlying symmetric encryption scheme SKE is correct. Therefore, every token session $\pi_T^i$ also has a unique sid.

This is an important observation, as it gives a guarantee that if a token session $\pi_T^i$ has a partner $\pi_C^j$, then $\pi_C^j$ is its *unique* partner.

After Game 8, it is also always the case that every token session where $bad_{11}$ can happen must have finished Bind passively, through an Execute query or a passive Send-Bind-T query. Indeed, after Game 8, $\mathcal{A}$ can no longer actively attack a token session $\pi_T^i$ via the Send-Bind-T oracle without corrupting the pin. However, if $\mathcal{A}$ does corrupt the pin, then she can never cause $bad_{11}$, i.e., can never win the game via a valid forgery against any session of token $T$.

Therefore, *only* passive executions of Bind can create token sessions where $bad_{11}$ can happen. This also gives us an additional guarantee that such sessions always have a binding state sampled uniformly at random, since these binding states were never checked for collisions in Game 9 because they are never transmitted to $\mathcal{A}$. [17]

**Reduction to the SUF-CMA security of MAC** To bound $\mathcal{A}$'s advantage against Game 11, we consider an adversary $\mathcal{B}_{11}$ simulating Game 11 and fix a value $n_{TS}$ for the maximum amount of token sessions that finished Bind passively. Then, $\mathcal{B}_{11}$ guesses at random in which token session $bad_{11}$ will happen first. We refer to that target session as $\pi_{T,target}^i$.

---

[17] This guarantee is especially significant when considering the proof for the original CTAP 2.1, since it allows to bound $\mathcal{A}$'s advantage by reducing to the SUF-CMA security of MAC.

If $\mathcal{A}$ ever queries Compromise on the client session that is partnered with $\pi^i_{T,\text{target}}$, then the simulation aborts, since it is now impossible to cause $\text{bad}_{11}$ on the target session $\pi^i_{T,\text{target}}$ (recall that if $\pi^j_C.\text{compromised} = \text{true}$, then $\mathcal{A}$ can never win via a valid forgery against its partner token session).

Whenever $\mathcal{A}$ queries the Auth-C oracle with a message $M$ such that $\pi^j_C$ is the partner of the target session $\pi^i_{T,\text{target}}$, $\mathcal{B}_{11}$ calls the $\mathcal{O}_{\text{Auth}}$ oracle to obtain a challenge tag $t$ for $M$, and returns $(M, t)$ to $\mathcal{A}$.

Whenever $\mathcal{A}$ queries the Validate-T oracle with message $M$ and tag $t$ such that $\pi^i_T$ is the target session $\pi^i_{T,\text{target}}$, $\mathcal{B}_{11}$ immediately returns $M$ and $t$ to its challenger and ends the simulation. If $\pi^i_T$ is not the target session, $\mathcal{B}_{11}$ continues simulating Game 11 until either $\mathcal{A}$ queries Validate-T for session $\pi^i_{T,\text{target}}$ or $\text{bad}_{11}$ happens.

It is not hard to see then, that if $\text{bad}_{11}$ ever happens in the target session $\pi^i_{T,\text{target}}$, $\mathcal{B}_{11}$ always wins the SUF-CMA game by returning the $(M, t)$ given to $\pi^i_{T,\text{target}}$ by $\mathcal{A}$ in a Validate-T query.

---

$\mathcal{B}_{11}(\mathcal{A}, \mathcal{O}_{\text{Auth}}, \mathcal{O}_{\text{Ver}})$:

$\quad \mathcal{L}_{\text{authC}}, \mathcal{L}_{\text{authT}} \leftarrow \emptyset, \mathcal{L}_{\text{valid}} \leftarrow \emptyset, \mathcal{L}_{\text{corrupt}} \leftarrow \emptyset$
$\quad \mathcal{L}_{\text{sCDH}}, \mathcal{L}_{\mathcal{H}_2} \leftarrow \emptyset$
$\quad \mathcal{L}_{\text{TokenSetup}} \leftarrow \emptyset$
$\quad \mathcal{L}_{\text{pk}} \leftarrow \emptyset$
$\quad \mathcal{L}_{\text{hash}} \leftarrow \emptyset$
$\quad \mathcal{L}_{\text{pin}} \leftarrow \emptyset, \mathcal{L}_{\text{setupKeys}} \leftarrow \emptyset$
$\quad \mathcal{L}_{\text{ct}} \leftarrow \emptyset$
$\quad \mathcal{L}_{\text{sampled}} \leftarrow \emptyset$
$\quad \mathcal{L}_{\text{pt}} \leftarrow \emptyset$
$\quad \text{bad}_{11} \leftarrow \text{false}$
$\quad \text{guess} \xleftarrow{\$} \{0, ..., n_{\text{TS}} - 1\}$
$\quad \mathcal{L}_{\text{TS}} \leftarrow \emptyset, (M_{\text{chal}}, t_{\text{chal}}) \leftarrow (\bot, \bot)$
$\quad \text{win-SUF-t} \leftarrow 0$
$\quad () \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(1^\lambda)$
$\quad \textbf{return } (M_{\text{chal}}, t_{\text{chal}})$

**Fig. 68.** Adversary $\mathcal{B}_{11}$'s simulation of game 11.

---

**Advantage measure** The advantage of $\mathcal{B}_{11}$ against the SUF-CMA security of MAC can be calculated as:

$$\text{Adv}^{\text{SUF-CMA}}_{\text{MAC}}(\mathcal{B}_{11}) \geq \frac{\text{Pr}_{11}[\text{bad}_{11}]}{n_{\text{TS}}} \tag{71}$$

$$n_{\text{TS}} \, \text{Adv}^{\text{SUF-CMA}}_{\text{MAC}}(\mathcal{B}_{11}) \geq \text{Pr}_{11}[\text{bad}_{11}] \tag{72}$$

Since games 10 and 11 are identical until $\text{bad}_{11}$, by the difference lemma, we have:

$$\text{Pr}_{11}[\text{bad}_{11}] \geq |\, \text{Pr}_{10} - \text{Pr}_{11} \,| \tag{73}$$

Thus, we have:

$$n_{\text{TS}} \, \text{Adv}^{\text{SUF-CMA}}_{\text{MAC}}(\mathcal{B}_{11}) \geq |\, \text{Pr}_{10} - \text{Pr}_{11} \,| \tag{74}$$

```
Auth-C (C, j, M):                                              Validate-T (T, i, M, t, d):
    if π_C^j =⊥ or π_C^j.st_exe ≠ bindDone or π_C^j.isValid = false then    if π_T^i =⊥ or π_T^i.st_exe ≠ bindDone or π_T^i.isValid = false then
        return ⊥                                                    return ⊥
    π_{T,target}^i ← L_TS[guess − 1]                           (C, j) ← tokenBindPartner(T, i)
    if π_{T,target}^i ≠⊥ ∧ (C, j) = tokenBindPartner(T, i) then    if π_T^i = L_TS[guess − 1] then
        (M, t) ← O_Auth(M)                                         (M_chal, t_chal) ← (M, t)
    else                                                           return
        (M, t) ←$ auth-C(π_C^j, M)                             else
    L_authC ← L_authC ∪ {(C, j, M, t)}                             status ← validate-T(π_T^i, M, t, d)
    return (M, t)                                              if (C, j, M, t) ∉ L_authC then
                                                                   if (C, j) = (⊥, ⊥) or π_C^j.compromised = false then
Compromise (C, j):                                                     if π_T^i.pinCorr = false then
    if π_C^j =⊥ or π_C^j.st_exe ≠ bindDone then                             if status = accepted then
        return ⊥                                                               bad_11 ← true
    π_{T,target}^i ← L_TS[guess − 1]                                           (M_chal, t_chal) ← (M, t)
    if π_{T,target}^i ≠⊥ ∧ (C, j) = tokenBindPartner(T, i) then                 return
        abort                                                              return rejected
    π_C^j.compromised = True                                  if status = accepted then
    return π_C^j.bs                                                win-SUF-t ← Token-Win-SUF-t(T, i, M, t, d)
                                                              return status
```

**Fig. 69.** Modified Auth-C, Validate-T and Compromise oracles simulated by $\mathsf{bad}_{11}$.

```
obtainPinUvAuthToken-T(π_T^i, puvProtocol, c, c_ph):             st_T.m ← 3, st_T.pinRetries ← pinRetriesMax
    if puvProtocol ∉ st_T.puvProtocolList ∨ st_T.pinRetries = 0   for all puvProtocol' ∈ st_T.puvProtocolList do
    then                                                              st_T.puvProtocol'.resetPuvToken()
        return (⊥, false)                                        π_T^i.bs ← π_T^i.puvProtocol.pt
    K ← st_T.puvProtocol.decapsulate(c)                          if active_key = true ∧ (st_T.puvProtocol.pk, c) ≠ L_setupKeys[T]
    if K =⊥ then                                                 then
        return (⊥, false)                                            if π_T^i.bs ∈ L_pt then
    st_T.pinRetries ← pinRetries − 1                                     abort
    (c', c'_ph, -, -) ← L_ct[-, π_T^i]                               L_pt ← L_pt ∪ {π_T^i.bs}
    active_key ← c' ≠ c                                              c_pt ←$ st_T.puvProtocol.encrypt(K, π_T^i.bs)
    active_{c_ph} ← c'_ph ≠ c_ph                                 else
    for all (π_{C'}^{j'}, π_{T'}^{i'}, c', c'_ph, -, -) ∈ L_ct s.t. T' = T do    p̃t ←$ {0, 1}^{len(π_T^i.bs)}
        if c'_ph = c_ph ∧ c' = c then                                if p̃t ∈ L_pt then
            active_{c_ph} ← false                                        abort
            active_key ← false                                       L_pt ← L_pt ∪ {p̃t}
    verifyPin ← active_key ∨ active_{c_ph}                           if π_T^i.bs ∈ L_{H_3} then
    if verifyPin = true then                                             abort
        pinHash ← st_T.puvProtocol.decrypt(K, c_ph)                  L_bs ← L_bs ∪ {π_T^i.bs}
        if L_corrupt[st_T.user] = true ∧ L_sampled[st_T.user] = false        c_pt ←$ st_T.puvProtocol.encrypt(K, p̃t)
        then                                                         L_ct[-, π_T^i] ← (c', c'_ph, c_pt, π_T^i.bs)
            pin ←$ P                                                  L_TS ← L_TS ∪ {π_T^i}
            L_valid[st_T.user] ← pin                             π_T^i.st_exe ← bindDone
            L_sampled[st_T.user] ← true                          π_T^i.canValidate ← true
        pinHash_T ← H_w(L_valid[st_T.user])                      π_T^i.sid ← π_T^i.sid || puvProtocol || c || c_ph || c_pt || false
        if L_corrupt[st_T.user] = false ∨ pinHash ≠ pinHash_T then    return (c_pt, false)
            st_T.puvProtocol.regenerate()
            st_T.m ← st_T.m − 1
            if st_T.m = 0 then
                authPowerUp-T(st_T)
                return (⊥, true)
            else
                return (⊥, false)
```

**Fig. 70.** obtainPinUvAuthToken-T function simulated by $\mathcal{B}_{11}$.

Since each token session can finish Bind passively only in Execute and Send-Bind-T queries, we know that $n_{\mathsf{TS}} \leq q_{\mathsf{E}} + q_{\mathsf{Send}}$. Therefore:

$$(q_{\mathsf{E}} + q_{\mathsf{Send}})\, \mathsf{Adv}^{\mathsf{SUF\text{-}CMA}}_{\mathsf{MAC}}(\mathcal{B}_{11}) \geq\ \mid \mathsf{Pr}_{10} - \mathsf{Pr}_{11} \mid \tag{75}$$

## Game 12

In this game, when $\mathcal{A}$ delivers a message $M$ and a tag $t$ to a client session $\pi_C^j$ by querying oracle Validate-C, the challenger always rejects, provided $M$ and $t$ were not output by $\pi_C^j$'s partner, $\pi_C^j$ was not compromised and the pin associated with $\pi_C^j$'s partner was not corrupted. In sum, given $M$ and $t$ that would result in $\mathcal{A}$ winning the experiment, the challenger rejects instead, making it impossible for $\mathcal{A}$ to win by forging a message tag and delivering it to any client session. We flag this event by creating a bad event $\mathsf{bad}_{12}$.

---

Game12($\mathcal{A}$):

  $\mathcal{L}_{\mathsf{authC}}, \mathcal{L}_{\mathsf{authT}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{valid}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{corrupt}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{sCDH}}, \mathcal{L}_{\mathcal{H}_2} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{TokenSetup}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{pk}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{hash}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{pin}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{setupKeys}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{ct}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{sampled}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{pt}} \leftarrow \emptyset$
  $\mathcal{L}_{\mathsf{bs}}, \mathcal{L}_{\mathcal{H}_3} \leftarrow \emptyset$
  $\mathsf{bad}_{11} \leftarrow \mathsf{false}$
  $\mathsf{bad}_{12} \leftarrow \mathsf{false}$
  $\mathsf{win\text{-}SUF\text{-}t} \leftarrow 0$
  $() \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(1^\lambda)$
  **return** win-SUF-t

Client-Win-SUF-t $(C, j, M, t)$:

  **if** $\exists (C_1, j_1), (C_2, j_2)$ s.t. $(C_1, j_1) \neq (C_2, j_2)$ **and** $\pi_{C_1}^{j_1}.\mathsf{st}_{\mathsf{exe}} = \pi_{C_2}^{j_2}.\mathsf{st}_{\mathsf{exe}} = \mathsf{bindDone}$
  **and** $\pi_{C_1}^{j_1}.\mathsf{sid} = \pi_{C_2}^{j_2}.\mathsf{sid}$ **then return** 1

  **if** $\exists (T_1, i_1), (T_2, i_2)$ s.t. $(T_1, i_1) \neq (T_2, i_2)$ **and** $\pi_{T_1}^{i_1}.\mathsf{st}_{\mathsf{exe}} = \pi_{T_2}^{i_2}.\mathsf{st}_{\mathsf{exe}} = \mathsf{bindDone}$
  **and** $\pi_{T_1}^{i_1}.\mathsf{sid} = \pi_{T_2}^{i_2}.\mathsf{sid}$ **then return** 1

  $(T, i) \leftarrow \mathsf{clientBindPartner}(C, j)$
  **if** $(T, i, M, t) \notin \mathcal{L}_{\mathsf{authT}}$ **then**
    **if** $\pi_C^j.\mathsf{compromised} = \mathsf{false}$ **then**
      **if** $\pi_T^i.\mathsf{pinCorr} = \mathsf{false}$ **then**
        **return** $1 \wedge \mathsf{bad}_{12} \neq \mathsf{true}$

  **return** 0

**Fig. 71.** Game 12 and function Client-Win-SUF-t modifications.

---

Validate-C $(C, j, M, t)$:

  **if** $\pi_C^j = \perp$ **or** $\pi_C^j.\mathsf{st}_{\mathsf{exe}} \neq \mathsf{bindDone}$ **or** $\pi_C^j.\mathsf{isValid} = \mathsf{false}$ **then**
    **return** $\perp$
  $(T, i) \leftarrow \mathsf{clientBindPartner}(C, j)$
  $\mathsf{status} \leftarrow \mathsf{validate\text{-}C}(\pi_C^j, M, t)$
  **if** $(T, i, M, t) \notin \mathcal{L}_{\mathsf{authT}}$ **then**
    **if** $\pi_T^i.\mathsf{compromised} = \mathsf{false}$ **then**
      **if** $\pi_C^j.\mathsf{pinCorr} = \mathsf{false}$ **then**
        **if** $\mathsf{status} = \mathsf{accepted}$ **then**
          $\mathsf{bad}_{12} \leftarrow \mathsf{true}$
          $\mathsf{win\text{-}SUF\text{-}t} \leftarrow \mathsf{Client\text{-}Win\text{-}SUF\text{-}t}(C, j, M, t)$
        **return** rejected
  **if** $\mathsf{status} = \mathsf{accepted}$ **then**
    $\mathsf{win\text{-}SUF\text{-}t} \leftarrow \mathsf{Client\text{-}Win\text{-}SUF\text{-}t}(C, j, M, t)$
  **return** status

Auth-T $(T, i, M)$:

  **if** $\pi_T^i = \perp$ **or** $\pi_T^i.\mathsf{st}_{\mathsf{exe}} \neq \mathsf{ValidateCmd}$ **or** $\pi_T^i.\mathsf{isValid} = \mathsf{false}$ **then**
    **return** $\perp$
  $(M, t) \xleftarrow{\$} \mathsf{auth\text{-}T}(\pi_T^i, M)$
  $\mathcal{L}_{\mathsf{authT}} \leftarrow \mathcal{L}_{\mathsf{authT}} \cup \{(T, i, M, t)\}$
  **return** $(M, t)$

**Fig. 72.** Modified Auth-T and Validate-C oracles - Game 12.

**Reduction to the SUF-CMA security of MAC** As was the case in Game 11, we bound the probability of $\mathcal{A}$ winning against Game 12 by reducing to the SUF-CMA security of the underlying MAC.

We consider an adversary $\mathcal{B}_{12}$ against the SUF-CMA game that simulates Game 12 and fix a value $n_{\mathsf{CS}}$ for the maximum amount of client sessions that finish Bind with a token session. Then, $\mathcal{B}_{12}$ guesses at random in which client session $\mathsf{bad}_{12}$ will happen first. We refer to that session as $\pi^{j}_{C,\mathsf{target}}$.

If $\mathcal{A}$ ever queries Compromise on $\pi^{j}_{C,\mathsf{target}}$, then the simulation aborts, since it is now impossible to cause $\mathsf{bad}_{12}$ on the target session $\pi^{j}_{C,\mathsf{target}}$ (recall that if $\pi^{j}_{C}.\mathsf{compromised} = \mathsf{true}$, then $\mathcal{A}$ can never win via a valid forgery against $\pi^{j}_{C}$).

Whenever $\mathcal{A}$ queries the Auth-T oracle with a message $M$ such that $\pi^{i}_{T}$ is the partner of the target session $\pi^{j}_{C,\mathsf{target}}$, $\mathcal{B}_{12}$ calls the $\mathcal{O}_{\mathsf{Auth}}$ oracle to obtain a challenge tag $t$ for $M$, and returns $(M, t)$ to $\mathcal{A}$.

Whenever $\mathcal{A}$ queries the Validate-C oracle with message $M$ and tag $t$ such that $\pi^{j}_{C}$ is the target session $\pi^{j}_{C,\mathsf{target}}$, $\mathcal{B}_{12}$ immediately returns $M$ and $t$ to its challenger and ends the simulation. If $\pi^{j}_{C}$ is not the target session, $\mathcal{B}_{12}$ continues simulating Game 12 until either $\mathcal{A}$ queries Validate-C for session $\pi^{j}_{C,\mathsf{target}}$ or $\mathsf{bad}_{12}$ happens.

It is not hard to see then, that if $\mathsf{bad}_{12}$ ever happens in the target session $\pi^{j}_{C,\mathsf{target}}$, $\mathcal{B}_{12}$ always wins the SUF-CMA game by returning the $(M, t)$ given to $\pi^{j}_{C,\mathsf{target}}$ by $\mathcal{A}$ in a Validate-C query.



$\mathcal{B}_{12}(\mathcal{A}, \mathcal{O}_{\mathsf{Auth}}, \mathcal{O}_{\mathsf{Ver}})$:
$\quad \mathcal{L}_{\mathsf{authC}}, \mathcal{L}_{\mathsf{authT}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{valid}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{corrupt}} \leftarrow \emptyset$
$\quad \mathcal{L}_{\mathsf{sCDH}}, \mathcal{L}_{\mathcal{H}_2} \leftarrow \emptyset$
$\quad \mathcal{L}_{\mathsf{TokenSetup}} \leftarrow \emptyset$
$\quad \mathcal{L}_{\mathsf{pk}} \leftarrow \emptyset$
$\quad \mathcal{L}_{\mathsf{pin}} \leftarrow \emptyset, \mathcal{L}_{\mathsf{setupKeys}} \leftarrow \emptyset$
$\quad \mathcal{L}_{\mathsf{ct}} \leftarrow \emptyset$
$\quad \mathcal{L}_{\mathsf{hash}} \leftarrow \emptyset$
$\quad \mathcal{L}_{\mathsf{sampled}} \leftarrow \emptyset$
$\quad \mathcal{L}_{\mathsf{pt}} \leftarrow \emptyset$
$\quad \mathsf{bad}_{11} \leftarrow \mathsf{false}$
$\quad \mathsf{bad}_{12} \leftarrow \mathsf{false}$
$\quad \mathsf{guess} \overset{\$}{\leftarrow} \{0, ..., n_{\mathsf{CS}} - 1\}$
$\quad \mathcal{L}_{\mathsf{CS}} \leftarrow \emptyset, (M_{\mathsf{chal}}, t_{\mathsf{chal}}) \leftarrow (\bot, \bot)$
$\quad \mathsf{win\text{-}SUF\text{-}t} \leftarrow 0$
$\quad () \overset{\$}{\leftarrow} \mathcal{A}^{\mathcal{O}}(1^{\lambda})$
$\quad \mathbf{return}\ (M_{\mathsf{chal}}, t_{\mathsf{chal}})$

Execute $(T, i, C, j)$:
$\quad \mathsf{pin}_U \leftarrow \mathcal{L}_{\mathsf{valid}}[\mathsf{st}_T.\mathsf{user}]$
$\quad \mathbf{if}\ \mathsf{st}_T = \bot\ \mathbf{or}\ \pi^i_T \neq \bot\ \mathbf{or}\ \pi^j_C \neq \bot\ \mathbf{or}\ \mathsf{pin}_U = \bot\ \mathbf{then}$
$\quad\quad \lfloor\ \mathbf{return}\ \bot$
$\quad \pi^i_T \leftarrow \mathsf{st}_T$
$\quad \mathsf{trans}, m_C, m_T \leftarrow \bot$
$\quad \mathcal{L}_{\mathsf{ct}}[\pi^j_C, \pi^i_T] \leftarrow (\bot, \bot, \bot)$
$\quad \mathbf{while}\ \pi^j_C.\mathsf{st}_{\mathsf{exe}} \neq \mathsf{bindDone}\ \mathbf{do}$
$\quad\quad m_T \overset{\$}{\leftarrow} \mathsf{Bind\text{-}T}(\pi^i_T, m_C)$
$\quad\quad m_C \overset{\$}{\leftarrow} \mathsf{Bind\text{-}C}(\pi^j_C, U, m_T)$
$\quad\quad \mathsf{trans} \leftarrow \mathsf{trans}\ ||\ m_T\ ||\ m_C$
$\quad \mathcal{L}_{\mathsf{CS}} \leftarrow \mathcal{L}_{\mathsf{CS}} \cup \{\pi^j_C\}$
$\quad \mathbf{for\ all}\ i' \neq i\ \mathbf{and}\ \pi^{i'}_T \neq \bot\ \mathbf{do}$
$\quad\quad \lfloor\ \pi^{i'}_T.\mathsf{isValid} \leftarrow \mathsf{false}$
$\quad \mathbf{return}\ \mathsf{trans}$

**Fig. 73.** Adversary $\mathcal{B}_{12}$'s simulation of game 12.

**Advantage measure** The advantage of $\mathcal{B}_{12}$ against the SUF-CMA security of MAC is upper bounded by:

$$\mathsf{Adv}^{\mathsf{SUF\text{-}CMA}}_{\mathsf{MAC}}(\mathcal{B}_{12}) \geq \frac{\mathsf{Pr}_{12}[\mathsf{bad}_{12}]}{n_{\mathsf{CS}}} \tag{76}$$

```
Validate-C (C, j, M, t):
    if π_C^j = ⊥ or π_C^j.st_exe ≠ bindDone or π_C^j.isValid = false then
        return ⊥
    (T, i) ← clientBindPartner(C, j)
    if π_C^j = L_CS[guess − 1] then
        (M_chal, t_chal) ← (M, t)
        return
    else
        status ← validate-C(π_C^j, M, t)
    if (T, i, M, t) ∉ L_authT then
        if π_T^i.compromised = false then
            if π_C^j.pinCorr = false then
                if status = accepted then
                    bad_12 ← true
                    (M_chal, t_chal) ← (M, t)
                    return
                return rejected
    if status = accepted then
        win-SUF-t ← Client-Win-SUF-t(C, j, M, t)
    return status

Auth-T (T, i, M):
    if π_T^i = ⊥ or π_T^i.st_exe ≠ bindDone or π_T^i.isValid = false then
        return ⊥
    π_{C,target}^j ← L_CS[guess − 1]
    if π_{C,target}^j ≠ ⊥ ∧ (T, i) = clientBindPartner(C, j) then
        (M, t) ← O_Auth(M)
    else
        (M, t) ←$ auth-T(π_T^i, M)
    L̄_authT ← L_authT ∪ {(T, i, M, t)}
    return (M, t)

Compromise (C, j):
    if π_C^j = ⊥ or π_C^j.st_exe ≠ bindDone then
        return ⊥
    if π_C^j = L_CS[guess − 1] then
        abort
    π_C^j.compromised = True
    return π_C^j.bs
```

**Fig. 74.** Modified Auth-T, Validate-C and Compromise oracles simulated by $\mathcal{B}_{12}$.

$$n_{CS} \; \mathsf{Adv}_{MAC}^{SUF\text{-}CMA}(\mathcal{B}_{12}) \geq \mathsf{Pr}_{12}[\mathsf{bad}_{12}] \tag{77}$$

Since games 11 and 12 are identical until $\mathsf{bad}_{12}$ happens, by the difference lemma, we have:

$$\mathsf{Pr}_{12}[\mathsf{bad}_{12}] \geq |\, \mathsf{Pr}_{11} - \mathsf{Pr}_{12} \,| \tag{78}$$

Thus, we have:

$$n_{CS} \; \mathsf{Adv}_{MAC}^{SUF\text{-}CMA}(\mathcal{B}_{12}) \geq |\, \mathsf{Pr}_{11} - \mathsf{Pr}_{12} \,| \tag{79}$$

Since each client session can finish Bind only in Execute queries, we know that $n_{CS} \leq q_E$. Therefore:

$$q_E \; \mathsf{Adv}_{MAC}^{SUF\text{-}CMA}(\mathcal{B}_{12}) \geq |\, \mathsf{Pr}_{11} - \mathsf{Pr}_{12} \,| \tag{80}$$

**Final Analysis**

Note that, at this point, $\mathcal{A}$ cannot win against Game 12, since it must be able to win via one of the four conditions in Token-Win-SUF-t or one of the three conditions in Client-Win-SUF-t.

**Win via Token-Win-SUF-t** Condition 1 of Token-Win-SUF-t ($d = 0$) is always false, since Token-Win-SUF-t is only called after a validation from the Validate-T oracle is successful, which is dependant on the user decision bit $d$.

Conditions 2 and 3 of Token-Win-SUF-t will also never be true. Indeed, since collisions between both client and token sid values have been ruled out in Game 9, it can never be the case that two different client or token sessions have the same sid. Thus, these conditions are always false.

Condition 4 of Token-Win-SUF-t is also always false. Recall that, for $\mathcal{A}$ to win, it must give a pair $(M, t)$ to the Validate-T oracle for a token session $\pi_T^i$ and $(M, t)$ must pass validation. Additionally, $\pi_T^i$ must not have had its pin corrupted, and either not have a partner or have a unique partner that was not compromised. In the latter case, $(M, t)$ must also not have been output by that partner.

Indeed, if the target $\pi_T^i$ which is validating $(M, t)$ from $\mathcal{A}$ has a (unique) partner $\pi_C^j$, then $\mathcal{A}$ hasn't actively attacked $\pi_T^i$ during the Bind session, and therefore has no information whatsoever about the pinToken, since the experiment no longer transmits the encrypted pinToken in passive Bind sessions.

If $\mathcal{A}$ actively attacked $\pi_T^i$ by performing a Bind session itself, in order to attempt to obtain the pinToken, the experiment always rejects, provided the pin was not corrupted. If it was corrupted, then $\mathcal{A}$ can never win against any session of that token. This eliminates the attack where $\mathcal{A}$ simply guesses the pin.

$\mathcal{A}$ can also never get any information whatsoever about the pin without corrupting the user $U$ to which it belongs, since no pin is ever sampled before $\mathcal{A}$ either corrupts the user or atempts to guess the pin for the first time by querying Send-Bind-T, and all ciphertexts $c_p$ and $c_{ph}$ transmitted by a client to a token have constant values.

Finally, $\mathcal{A}$ can never win by correctly forging a tag $t$, since the token session always rejects any valid forgery attempt from $\mathcal{A}$.

**Win via Client-Win-SUF-t** Conditions 1 and 2 are identical to conditions 2 and 3 from Token-Win-SUF-t, so they are always false.

Condition 3, similarly to condition 4 of Token-Win-SUF-t, is also false, since $\mathcal{A}$ can never learn anything about the pinToken used to authenticate messages to any client session without corrupting a pin or compromising the session itself, and can never win by actually submitting a valid forged tag $t$ because the client session always rejects such an attempt.

**Advantage measure** Given that $\mathcal{A}$ can no longer win against Game 12, we have:

$$\mathsf{Pr}_{12} = 0 \tag{81}$$

Note that the running times of all adversaries are close to that of $\mathcal{A}$.

This completes the proof.