Token meets Wallet: Formalizing Privacy and Revocation for FIDO2

Lucjan Hanzlik
hanzlik@cispa.de
CISPA Helmholtz Center for Information Security

Julian Loss
loss@cispa.de
CISPA Helmholtz Center for Information Security

Benedikt Wagner
benedikt.wagner@cispa.de
CISPA Helmholtz Center for Information Security

ABSTRACT

The FIDO2 standard is widely-used class of challenge-response type protocols that allows to authenticate to an online service using a hardware token. Barbosa et al. (CRYPTO ‘21) provided the first formal security model and analysis for the FIDO2 standard. However, their model has two shortcomings: (1) it does not include privacy, one of the key features claimed by FIDO2 (2) their model and proofs apply only to tokens that store all secret keys locally. In contrast, due to limited memory, most existing FIDO2 tokens use one of the following approaches to handle an unlimited number of keys. *Key derivation* derives a fresh per-server secret key from a common seed. *Key wrapping* stores an encryption of the key on the server and retrieves them for each authentication. These approaches substantially complicate the protocols and their security analysis. In particular, they bear additional risks for privacy and security of FIDO2 that are not captured in the model Barbosa et al. model.

In this paper, we revisit the security of the FIDO2 as implemented in practice. Our contributions are as follows. (1) We adapt the model of Barbosa et al. so as to capture authentication tokens using key derivation or key wrapping. (2) In our adapted model, we provide the *first formal definition of privacy for FIDO2* and show that these common FIDO2 token implementations are secure in our model, if the underlying building blocks are chosen appropriately. (3) Finally, we address the unsolved problem of *global key revocation* in FIDO2. We first provide appropriate syntax of a revocation procedure and extend our model to support this feature. We then provide the first secure global key revocation protocol for FIDO2. Our solution is based on the popular BIP32 standard used in cryptocurrency wallets.

CCS CONCEPTS

• **Security and privacy** → **Mathematical foundations of cryptography**; **Security protocols**;

KEYWORDS

FIDO2; BIP32; unlinkability; revocation;

1 INTRODUCTION

Online authentication is one of the most pressing challenges faced by security engineers and cryptographers today. Reliable authentication is an important concern for both the security of user’s accounts as well as the reputation of service providers. A simple way to strengthen the security of an authentication process is to introduce additional authentication factors. Usually, the user has to just provide a login and password (*something she knows*). A popular way to introduce a second factor is for the user to use a device (*something she has*). This is usually implemented by registering a device to the user account.

Universal Second Factor (U2F) (or CTAP1, as it is currently named) is a protocol to achieve two-factor authentication using a designated device. This cryptographic hardware token (also called *authenticator*) runs a simple piece of code and interacts with the user’s platform (also called *client* or *agent*) represented, e.g., by a web browser. The agent acts as a proxy device during the authentication process between an authenticator and a server (also called *relying party*). The main benefit of this solution over the login/password approach is the protection against phishing attacks and database breaches.

CTAP1/U2F and its successor CTAP2 are part of the FIDO authentication specification [2]. Together with the complement W3C web authentication [3] they form the state-of-the-art for online authentication using security tokens.

In recent work, Barbosa et al. [6] gave the first formal model for token-based two-factor authentication and provided a security proof for the FIDO2 standard. On one hand, their model provides an important starting point for further exploration of FIDO2’s security properties. On the other hand, it does not accurately model several key aspects of FIDO2 as used in practice. In this work, we revisit the FIDO2 standard and give a more complete security analysis of its security features. We also augment the existing standard with a new feature that allows to easily revoke keys of compromised tokens.

1.1 Our Contribution

We begin by identifying several disparities between the model of Barbosa et al. and how FIDO2 is commonly used in practice, which we explain below. For each of these points, we explain how our work amends the existing security model to close these gaps. We also discuss a practical feature for key revocation that we call *global revocation.* To the best of our knowledge, this property of FIDO2 has not been discussed or formalized prior to this work.

Attestation. Barbosa et al. consider a model where the authenticator and the relying party are initially provided a pair of attestation secret key and attestation public key by the (trusted) manufacturer. While attestation is common practice for critical interactions such as banking transactions, it is rarely used for more mundane scenarios such as logging in to an internet account. Also, naive attestation violates privacy [23]. Therefore, to reason about the security and privacy of FIDO2 in its most widely used form, we consider a model *without attestation.*

Resident Keys vs. Privacy. FIDO2 can be used in two different modes. The first is to store the secret keys of the authenticator locally, i.e., on the authentication token. Such keys are known as *resident keys*. In practice, a more common way is to store (encrypted) keys with the server, in the form of so-called *non-resident keys*. This technique is also called *key wrapping*. Another approach to non-resident keys is to use a key derivation function. Since the token is often limited with regards to storage space, non-resident keys provide a distinct storage advantage over resident keys.

On the downside, non-resident keys give the server additional information that it may use to link separate sessions of the token. This violates the privacy properties claimed by many common implementations of FIDO2.

Let us illustrate this using a (pathological) implementation of key wrapping. Assume that the server stores ciphertexts of the form $(\mathsf{Enc}(k, \mathsf{sk}), H(k))$, where Enc is a symmetric encryption scheme, H is a hash function, k denotes the encryption key, and sk denotes the signing key used for authentication. Observe that the above scheme appends a hash of the secret encryption key to the ciphertext of Enc. It is not hard to see that if H is modelled as a random oracle with independent randomness of Enc, the above scheme remains secure. However, ciphertexts produced by the same token (i.e., using the same key k) can trivially be linked using the second component. Interestingly, we observe that the Yubico implementation of FIDO2 [32] uses an authenticated encryption scheme based on CCM-mode of encryption. To the best of our knowledge, key anonymity (i.e., unlinkability of ciphertexts produced with the same encryption key) of this mode of encryption has not been formally explored.

The above discussion shows that the use of non-resident keys leads to unexpected subtleties with regards to privacy across different sessions of the same token. Unfortunately, as Barbosa et al. consider exclusively the setting of resident keys, their model does capture privacy gurantees. One of our key contributions is to augment their model with a suitable notion of privacy.

Adding Global Revocation. When using FIDO2, a crucial aspect is how to securely revoke keys in case access to the authentication token is ever lost. Indeed, the usability study by Lyastani et al. [26] shows that one of the top concerns of users is that an adversary can gain access to their account in case their FIDO2 authenticator gets lost or is stolen. In such a case, we desire a public procedure that allows to simultaneously revoke *all keys* associated with such an authenticator. We stress that this includes also any future keys that the (compromised) authenticator may attempt to register with the relying party. We refer to such a procedure as *global revocation*. Despite significant benefits, current authenticator implementations do not support global revocation.

Implementing the key generation process on the authenticator in a way that is compatible with global revocation is challenging. To see the issue, suppose that the authenticator naively generates each key pair using a fresh random seed. In this case, it is not possible to revoke any of the future keys that the authenticator could derive post-compromise. Thus, any solution for global revocation seems to require that the authenticator generates keys using correlated randomness (e.g., deterministically from the same seed). Later, it can reveal this randomness to revoke all past and future keys at once. However, this solution might introduce different issues: first, is it possible to link sessions if keys are correlated? Second, does FIDO2 remain secure against impersonation if the random seed is ever leaked?

Crypto Wallets to the Rescue. Fortunately, we can rely on a practical solution from the cryptocurrency space to solve this issue. A common approach to store a multitude of keys compactly is to use *deterministic key derivation*[1]. Considering the case of BIP32, the most widely implemented procedure for deterministic key derivation, keys are derived from a pair of master keys $\mathsf{msk} \in \mathbb{Z}_p, \mathsf{mpk} = g^{\mathsf{msk}}$ and a *chaincode* chain. Here, g denotes the base point of an elliptic curve and chain can be thought of as a random seed. To derive a fresh key pair for an identity id, BIP32 first computes $w = \mathsf{Hash}(\mathsf{id}, \mathsf{chain}, \mathsf{mpk})$, then sets $\mathsf{sk}_{\mathsf{id}} := \mathsf{msk} + w, \mathsf{pk}_{\mathsf{id}} := \mathsf{mpk} \cdot g^{w}$. Proving that ECDSA signatures remain unforgeable with respect to keys derived in this fashion once chain is leaked turns out be non-trivial. In recent work, Das et al. were the first to consider this stronger form of unforgeability and showed that it holds given that no message is ever signed twice [14]. On the other hand, the above procedure provides a very simple means of global revocation. To revoke all public keys associated with a pair msk, mpk, all one needs to do is to publish chain. Now, each relying party can revoke the appropriate public keys by deterministically recomputing them (for suitable identities) from mpk and chain. Given the widespread use of BIP32, our solution offers a practical means of global key revocation that can be directly implemented using a multitude of existing hardware devices.

1.2 Related Work

The first formal model for the FIDO2 authentication protocol was proposed by Barbosa et al. [6]. The authors introduced the notion of passwordless authentication that models the Webauthn protocol in the scenario when the token uses resident keys. They also introduce a security model for PIN-based access control that tries to formally define the CTAP protocol executed between the token and the client. Barbosa et al. show that CTAP only provides a weaker notion of access control and propose an alternative protocol based on a PAKE.

Guirat et al. [23] analyzed the Webauthn protocol using automated verification and modeled a simple privacy definition. They showed that if the same key is used for attestation it allows a server to link the same token. Feng et al. [20] also used automated verification to check the FIDO UAF protocol. Potential security and privacy issues that arise during the development phase of FIDO components were discussed by Alam et al. [4].

Another line of related work provides alternative protocols and usability studies. Chakraborty et al. [11] used a TPM implementation (simTPM) based on a sim card, as a secure and convenient FIDO2 authenticator. The problem of backdoored FIDO tokens was discussed Dauterman et al. [16]. They introduced an alternative design for a token, which is resistant against backdoors introduced to the device by the manufacturer. Frymann et al. [22] discussed the token backup procedure proposed by Yubico. A usability and acceptability study of FIDO2 was done by Lyastani et al. [26]. Florian et al. [19] show that despite security benefits, hardware tokens face acceptability challenges in small companies.

From a cryptographic point of view, FIDO2 is a simple authentication scheme, a primitive which has been extensively studied.

[1] Such a mechanism is colloquially referred to as a *wallet*.

Everything started with simple solutions. One of the first and well-known protocols is the Schnorr identification scheme [29]. Authenticated key exchange (AKE) protocols [17, 24, 25] simultaneously to authentication provide means to agree on a secret key between the interacting parties. A different approach to authentication is the folklore challenge-response protocol based on signature schemes, which FIDO2 is based on. This technique provides a framework for creating authentication protocols. Instantiated with group signatures [7, 12] the protocol provides a way for group members to anonymously authenticate to a server. A direct solution introduced by Teranishi, Furukawa, and Sako is anonymous authentication [10, 30] .

Dwork, Naor and Sahai [18] introduced the concept of deniable authentication where an interaction between the parties cannot be used as undeniable proof of interaction. This notion was later extended to proof systems [27]. An alternative approach to authentication using a hardware device is the extended access control protocol (EAC) [9, 13] for machine-readable travel documents (e.g. e-Passports). This protocol is used for local authentication can be easily be used to authenticate the document online.

2 PRELIMINARIES

In this section we fix some notation and provide the necessary cryptographic background that will be needed for our analysis.

Notation. We denote by $z \leftarrow \mathcal{A}(x)$ the execution of algorithm $\mathcal{A}$ on input x and with output z. We write $y \in \mathcal{A}(x)$ to indicate that y is a possible output of $\mathcal{A}$ on input x. By $r \xleftarrow{\$} S$ we mean that r is chosen uniformly at random over the set S. We will use $[n]$ to denote the set $\{1, \ldots, n\}$. Throughout the paper, we assume public parameters param are given implicitly to all algorithms.

Signatures. We recall the standard notion of digital signatures. We also introduce the notion of rerandomizable signature schemes [15, 21] and the corresponding security notion.

Definition 2.1 (Digital Signature Scheme.). A *digital signature scheme* is a tuple of algorithms $\mathsf{sig} = (\mathsf{kg}, \mathsf{sign}, \mathsf{ver})$ with the following properties.

- The randomized *key generation algorithm* kg takes as input parameters param. It outputs a secret key and public key $(\mathsf{sk}, \mathsf{pk})$.
- The randomized *signing algorithm* sign takes as input a secret key sk and a message m. It outputs a signature σ.
- The deterministic *verification algorithm* ver takes as input a public key pk a signature σ, and a message m. It outputs 0 (reject) or 1 (accept).

We say that a digital signature scheme is *correct* if for all $(\mathsf{sk}, \mathsf{pk}) \in \mathsf{kg}(\mathsf{param})$ and all messages $m \in \{0,1\}^*$ we have

$$\Pr_{\sigma \leftarrow \mathsf{sign}(\mathsf{sk}, m)}[\mathsf{ver}(\mathsf{pk}, \sigma, m) = 1] = 1.$$

Further, without loss of generality, we assume that secret keys are chosen uniformly at random and there is an algorithm toPK that maps secret keys to public keys, i.e. $\mathsf{pk} := \mathsf{toPK}(\mathsf{sk})$.

Definition 2.2 (Unforgeability under Chosen Message Attacks.). Let $\mathsf{sig} = (\mathsf{kg}, \mathsf{sign}, \mathsf{ver})$ be a digital signature scheme and consider the experiment $\text{euf-cma}_{\mathsf{sig}}^{\mathcal{A}}$ defined as follows:

- **Setup.** The experiment generates $(\mathsf{sk}, \mathsf{pk})$ via $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{kg}(\mathsf{param})$. It runs the adversary $\mathcal{A}$ on input pk.
- **Online Phase.** In this phase, $\mathcal{A}$ is given access to oracle Sign, which takes as input a message m and returns the signature $\sigma \leftarrow \mathsf{sign}(\mathsf{sk}, m)$.
- **Output Phase.** When $\mathcal{A}$ returns (m^*, σ^*), the experiment returns 1 if $\mathsf{ver}(\mathsf{pk}, \sigma^*, m^*) = 1$ and m^* was not queried to Sign. Otherwise, it returns 0.

We define the advantage of $\mathcal{A}$ in $\text{euf-cma}_{\mathsf{sig}}^{\mathcal{A}}$ as

$$\mathbf{Adv}_{\text{euf-cma},\mathsf{sig}}^{\mathcal{A}} := \Pr[\text{euf-cma}_{\mathsf{sig}}^{\mathcal{A}} = 1].$$

Definition 2.3 (Rerandomizable Signature Scheme.). A *rerandomizable signature scheme* is a tuple of algorithms $\mathsf{sig} = (\mathsf{kg}, \mathsf{sign}, \mathsf{srerand}, \mathsf{prerand}, \mathsf{ver})$, where $(\mathsf{kg}, \mathsf{sign}, \mathsf{ver})$ is a digital signature scheme, and algorithms srerand, prerand have the following properties.

- The deterministic *secret key rerandomization algorithm* srerand takes as input a secret key sk and a string ρ. It outputs a rerandomized key sk_ρ.
- The deterministic *public key rerandomization algorithm* prerand takes as input a public key pk and a string ρ. It outputs a rerandomized key pk_ρ.

Moreover, we require that for all $(\mathsf{sk}, \mathsf{pk}) \in \mathsf{kg}(\mathsf{param})$, key pairs $(\mathsf{sk}_\rho, \mathsf{pk}_\rho)$ generated as $\rho \leftarrow \{0,1\}^\lambda$, $\mathsf{sk}_\rho := \mathsf{srerand}(\mathsf{sk}, \rho)$, and $\mathsf{pk}_\rho := \mathsf{prerand}(\mathsf{pk}, \rho)$ are identically distributed to key pairs generated via $(\mathsf{sk}', \mathsf{pk}') \leftarrow \mathsf{kg}(\mathsf{param})$.

We next recall the security notion of unforgeability under honestly rerandomized keys [14, 15].

Definition 2.4 (Unforgeability under Honestly Rerandomized Keys.). Let $\mathsf{sig} = (\mathsf{kg}, \mathsf{sign}, \mathsf{srerand}, \mathsf{prerand}, \mathsf{ver})$ be a rerandomizable signature scheme and consider the experiment $\textbf{ufcma-hrk}1_{\mathsf{sig}}^{\mathcal{A}}$ defined as follows:

- **Setup.** The experiment generates $(\mathsf{sk}, \mathsf{pk})$ via $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{kg}(\mathsf{param})$. It runs the adversary $\mathcal{A}$ on input pk.
- **Online Phase.** In this phase, $\mathcal{A}$ is given access to oracles Sign and Rand:
  - Rand takes no inputs and returns $\rho \leftarrow \{0,1\}^\lambda$.
  - Sign takes as input a message m and a string $\rho \in \{0,1\}^\lambda$. If ρ was not a previous output of Rand or the pair (m, ρ) has been queried before, Sign returns $\perp$. Otherwise, it returns the signature σ computed via $\sigma \leftarrow \mathsf{sign}(\mathsf{srerand}(\mathsf{sk}, \rho), m)$.
- **Output Phase.** When $\mathcal{A}$ returns (σ^*, ρ^*, m^*), the experiment returns 1 if $\mathsf{ver}(\mathsf{prerand}(\mathsf{pk}, \rho^*), \sigma^*, m^*) = 1$, ρ^* was a previous output of Sign, and (m^*, ρ^*) was not queried to Sign. Otherwise, it returns 0.

We define the advantage of $\mathcal{A}$ in $\textbf{ufcma-hrk}1_{\mathsf{sig}}^{\mathcal{A}}$ as

$$\mathbf{Adv}_{\textbf{ufcma-hrk}1,\mathsf{sig}}^{\mathcal{A}} := \Pr[\textbf{ufcma-hrk}1_{\mathsf{sig}}^{\mathcal{A}} = 1].$$

Symmetric Key Encryption. Here, we recall the definition of symmetric key encryption that is both authenticated and anonymous. For the definition of security, we follow [5]. We note that a scheme that satisfies this notion can be constructed using for example the OCB mode of operation [28].

Definition 2.5 (Symmetric Key Encryption Scheme.). A *symmetric key encryption scheme* (SKE) is a tuple of algorithms $\mathsf{ske} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with the following properties.

- The randomized *key generation algorithm* Gen takes as input parameters param. It outputs a secret key sk.
- The randomized *encryption algorithm* Enc takes as input a secret key sk and a message m. It outputs a ciphertext c.
- The deterministic *decryption algorithm* Dec takes as input a secret key sk and a ciphertext c. It outputs either $\perp$ or a message m.

Definition 2.6 (Authenticated Anonymous Security for SKE.). Let $\mathsf{ske} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a symmetric key encryption scheme. For an algorithm $\mathcal{A}$ and a bit $b \in \{0, 1\}$, consider the following experiment $\mathbf{anon\text{-}auth}^{\mathcal{A}}_{\mathsf{ske},b}$:

- **Setup.** The experiment generates a key $\mathsf{sk} \leftarrow \mathsf{Gen}(\mathsf{param})$ and initializes a map $L[\cdot]$.
- **Online Phase.** The adversary $\mathcal{A}$ is run on input param with oracle access to oracles $\mathsf{EncO}_b, \mathsf{DecO}_b$, which are defined as follows.
  - $\mathsf{EncO}_0(m)$: Return $c \leftarrow \mathsf{Enc}(\mathsf{sk}, m)$.
  - $\mathsf{DecO}_0(c)$: Return $m := \mathsf{Dec}(\mathsf{sk}, c)$.
  - $\mathsf{EncO}_1(m)$: Sample a ciphertext c uniformly at random, define $L[c] := m$ and return c.
  - $\mathsf{DecO}_1(c)$: If $L[c] \neq \perp$, return $L[c]$. Otherwise, return $\perp$.
- **Output Phase.** The adversary outputs a bit b'. The experiment outputs b'.

We define the advantage of $\mathcal{A}$ against authenticated anonymous security of ske as

$$\mathbf{Adv}^{\mathcal{A}}_{\mathbf{anon\text{-}auth},\mathsf{ske}} := \left|\Pr[\mathbf{anon\text{-}auth}^{\mathcal{A}}_{\mathsf{ske},0} = 1] - \Pr[\mathbf{anon\text{-}auth}^{\mathcal{A}}_{\mathsf{ske},1} = 1]\right|.$$

3 MODELING THE FIDO2 PROTOCOL

We begin this section with an informal description of the FIDO2 standard. It can be used as means of passwordless authentication or as a second-factor for the standard login-via-password scenario.

The FIDO2 protocol template consists of a registration and an authentication process executed between a token (also called *authenticator*) and a server (also called *relying party*). This communication is relayed through a client (also called *agent*) which is implemented e.g. by the user's browser. More precisely, the user connects to the server via client interface and simultaneously interacts with the token. Below we provide a simplified description of the protocol. A schematic overview can be found in Figure 1.

Registration. The main purpose of the registration protocol is to bind the token to the user's account on the server. During this step, the server sends a unique identifier id_S (which is based on the server's domain name e.g. `login.example.com`) and a challenge rs to the client. This data is then processed and sent to the token which generates a server-specific key pair $(\mathsf{sk}, \mathsf{pk})$ and sends pk to the server via the client. The server stores this key and additional information in a *credential* cred.

Non-Resident Keys. To accommodate for memory-constraints, tokens commonly implement one of two techniques for outsourcing secret key storage that we describe below. Keys that are not stored on the token are called *non-resident keys*. By contrast, *resident keys* are kept internally by the token.

The first technique to outsource key storage is referred to as *key wrapping*. The server-specific secret key sk is encrypted and the resulting ciphertext cid (the *credential identifier*) is stored on the server. During authentication, cid is returned by the server and allows the token to restore sk. The server uses a structure known as a *key handle* to send cid to the token for authentications.

The second technique uses a *key derivation function* to compute the server-specific key pair using a master secret, id_S and a random seed cid chosen by the token during registration. Similarly to key wrapping, cid is stored (with no encryption necessary) on the server and can be retrieved for later authentication via the key handle structure.

Attestation. If required by the server, the public key pk and other data can also be *attested* by the token. In the following, we explain two ways to implement attestation.

The first option is to create an attestation signature using a fresh key pair. This approach requires the token to receive a certificate for each key pair. To receive such a certificate, the token must authenticate itself to the certification authority (also called *privacy CA*) using a special endorsement key, which is loaded during manufacturing. Note that the same key cannot be used to attest public keys for different services since this would immediately break unlinkability of separate user sessions with the server. Attestation, in this case, consists of a signature and an attestation certificate for the signing key.

The second option is the elliptic curve direct anonymous attestation scheme (ECDAA) [1] where an interaction with a privacy CA is not needed. Instead, the token receives a randomizable certificate on a secret signing key, e.g. by the manufacturer. To verify an attestation signature, the server is only required to use a public key provided by the CA. In other words, besides the authenticity of the attestation, the signature does not leak the signer's identity *but only that the signer is certified.* An advantage of ECDAA is that there is no need for an attestation certificate as with a privacy CA.

We note that the model and analysis in [6] focuses on attestation with a fixed key instead of the unlinkable attestation options discussed above.

Authentication. The authentication process begins with the server sending the following data to the client: the key handle generated during registration, including cid, the server's identifier id_S, and a challenge rs for the token to sign. The client processes this data, adds extensions (e.g. information about the `https` connection) and forwards it to the token. The token restores the server-specific signing key sk, signs the received data, and returns the signature to the server which accepts if the signature is valid. This signature is also called an *assertion*. An optional way to prevent physical cloning attacks is to implement a counter mechanism on the token.

3.1 Passwordless Authentication

We model passwordless authentication for the non-resident key case as follows. Similar to [6], in our model we consider a set of

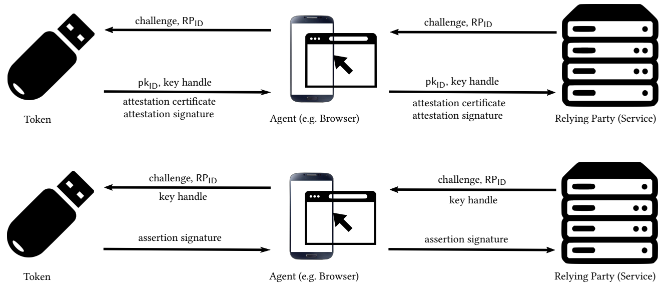

Figure 1: Registration (top) and authentication (bottom) in the FIDO2 authentication process (simplified).

parties $\mathcal{P} = \mathcal{T} \cup \mathcal{S}$, which is partitioned into the set of tokens $\mathcal{T}$ and the set of servers $\mathcal{S}$. Each party keeps an internal state. We do not model clients explicitly. Instead, we allow the adversary to do the computation that the clients do.

The state of servers $S \in \mathcal{S}$ is initially empty, while the state of tokens will be initialized with a long-term key msk. Also, we assume that each server has a unique identifier id_S. In all experiments that we define, we assume that these identifiers are given.

Syntax. Now, we define the syntax of our schemes. The syntax follows the setup introduced above.

Definition 3.1 (PlANRK). A *passwordless authentication scheme with non-resident keys* (PlANRK) is a tuple of algorithms PlANRK = (Gen, Reg, Auth) with the following properties:

- The randomized *key generation algorithm* Gen takes as input parameters par. It outputs a master secret key msk.
- The *registration protocol* Reg given as a tuple of stateful algorithms (rchallenge, rcommand, rresponse, rcheck).
  - The randomized *registration challenge generation algorithm* rchallenge takes as input a server identity id_S and outputs a challenge value c.
  - The deterministic *registration command creation algorithm* rcommand takes as input a server identity id_S and a challenge value c and outputs a message M_r.
  - The randomized *registration response algorithm* rresponse takes as input a master secret key msk, a server identity id_S and a message M_r and outputs and credential identifier cid and a response R_r.
  - The deterministic *registration check algorithm* rcheck takes as input a credential identifier cid and a response R_r and outputs a bit $b \in \{0, 1\}$ and a credential cred.
- The *authentication protocol* Auth given as a tuple of stateful algorithms (achallenge, acommand, aresponse, acheck).
  - The randomized *authentication challenge generation algorithm* achallenge takes as input a server identity id_S and outputs a challenge value c.
  - The deterministic *authentication command creation algorithm* acommand takes as input a server identity id_S and a challenge value c and outputs a message M_a.
  - The randomized *authentication response algorithm* aresponse takes as input a master secret key msk, a server identity id_S and a message M_a and outputs a response R_a.
  - The deterministic *authentication check algorithm* acheck takes as input a credential identifier cid and a response R_a and outputs a bit $b \in \{0, 1\}$

Algorithms rchallenge, rcheck, achallenge, acheck are executed by servers $S \in \mathcal{S}$, algorithms rcommand, acommand are executed by clients (which are not explicitly modeled in security definitions), and the corresponding algorithms rresponse, aresponse are executed by tokens $T \in \mathcal{T}$.

Definition 3.2 (Completeness of PlANRK). We say that a PlANRK PlANRK = (Gen, Reg, Auth) with Reg = (rchallenge, rcommand, rresponse, rcheck) and Auth = (achallenge, acommand, aresponse, acheck) is complete, if for all msk ∈ Gen(par), parties T and S, states $\mathsf{St}_T, \mathsf{St}_S$ the following experiment outputs 1:

(1) Initialize the state of party T with msk and St_T, and the state of party S with St_S.
(2) Run the registration protocol Reg of T at S, as follows:

$$c \leftarrow \mathsf{rchallenge}(\mathsf{id}_S),$$
$$M_r \leftarrow \mathsf{rcommand}(\mathsf{id}_S, c),\ (\mathsf{cid}, R_r) \leftarrow \mathsf{rresponse}(\mathsf{msk}, \mathsf{id}_S, M_r),$$
$$(b_r, \mathsf{cred}) \leftarrow \mathsf{rcheck}(\mathsf{cid}, R_r).$$

(3) If $b_r = 0$, output 0. Otherwise, run the authentication protocol Auth of T at S, which is as follows:

$$c \leftarrow \mathsf{achallenge}(\mathsf{id}_S), \qquad M_a \leftarrow \mathsf{acommand}(\mathsf{id}_S, c),$$
$$R_a \leftarrow \mathsf{aresponse}(\mathsf{msk}, \mathsf{id}_S, \mathsf{cid}, M_a), \qquad b_a \leftarrow \mathsf{acheck}(\mathsf{cid}, R_a).$$

(4) Return b_a.

Security. Next, we focus on defining security notions for PlANRK. To do so, we first define server and token oracles an adversary may access. These oracles model the capability of an adversary to freely communicate with tokens and servers.

Definition 3.3 (Server and Token Oracles). Let $\mathcal{A}$ be an algorithm and PlANRK = (Gen, Reg, Auth) be a PlANRK. We associate each party $P \in \mathcal{T} \cup \mathcal{S}$ with a set of oracles $\pi_P^{i,j}$ which model two types of instances corresponding to registration and authentication. Each party is represented by a number of instances with a shared state and we will use the superscript i to indicate the particular instance of the party. In other words, we we refer to $\pi_P^{i,j}$ for $j = 0$ as the i-th registration instance of party P and for $j \geq 1$ as the j-th authentication instance of P corresponding to the i-th registration. The oracles can be accessed by $\mathcal{A}$ via interfaces Start, Challenge, Complete as follows:

- $\mathsf{Start}(\pi_S^{i,j})$: Server oracle $\pi_S^{i,j}$ executes $\mathsf{rchallenge}(\mathsf{id}_S)$ in case $j = 0$ or $\mathsf{achallenge}(\mathsf{id}_S)$ in case $j > 0$. The result is returned to $\mathcal{A}$.
- $\mathsf{Challenge}(\pi_T^{i,j}, \mathsf{id}_S, \mathsf{cid}, M)$: Token oracle $\pi_T^{i,j}$ runs algorithm $\mathsf{rresponse}(\mathsf{id}_S, M)$ (if $j = 0$) or $\mathsf{aresponse}(\mathsf{id}_S, \mathsf{cid}, M)$ (if $j > 0$). The result is returned to $\mathcal{A}$.
- $\mathsf{Complete}(\pi_S^{i,j}, \mathsf{cid}, R)$: Server oracle $\pi_S^{i,j}$ runs $\mathsf{rcheck}(\mathsf{cid}, R)$ (if $j = 0$) or $\mathsf{acheck}(\mathsf{cid}, R)$ (if $j > 0$). The result is returned to $\mathcal{A}$.

We assume without loss of generality that an oracle is only executed once with the same type of query. Note that all oracles take as input instances token or server instances and thus $\mathcal{A}$ can just query the same input with the next instance $j' = j + 1$ of the target party.

We follow the work of Bellare et al. [8] to define partnering of oracles, which will be used in the winning condition of our security experiments. Informally, two oracles $\pi_S^{i,j}$ and $\pi_T^{i',j'}$ are partnered, if they share the same view. Here, the notion of view must be defined by the protocol. Intuitively, one should think of partnered oracles as if the adversary just forwarded messages between these oracles.

Definition 3.4 (Session Identifiers and Partnering). Let $\mathsf{PlANRK} = (\mathsf{Gen}, \mathsf{Reg}, \mathsf{Auth})$ be a PlANRK and consider the oracles from Definition 3.3. Let V_t be a function that takes as input the transcript $tr_T^{i,j} = (\mathsf{id}_S, \mathsf{cid}, M, R)$ that a token $T \in \mathcal{T}$ observes in an oracle call to $\mathsf{Challenge}(\pi_T^{i,j}, \cdot)$, and outputs a bitstring $V_t(tr_T^{i,j})$. Similarly, let V_s be a function that takes as input the transcript $tr_S^{i,j} = (c, \mathsf{cid}, R)$ that a server $S \in \mathcal{S}$ observes in oracle calls to $\mathsf{Start}(\pi_S^{i,j})$, $\mathsf{Complete}(\pi_S^{i,j}, \cdot)$, and outputs a bitstring $V_s(tr_S^{i,j})$. We assume that these functions are specified by PlANRK.

We say that oracles $\pi_T^{i,j}$ and $\pi_S^{i',j'}$ are partnered, if the following hold:

$$(j = 0 \iff j' = 0) \land V_t(tr_T^{i,j}) = V_s(tr_S^{i',j'}).$$

We now define what it means for a passwordless authentication protocol with non-resident keys to be secure against impersonation. Informally, we say that if this property holds then the token must be used to authenticate against a server and a single interaction cannot be used to authenticate multiple times. More formally, we define impersonation security as follows.

Definition 3.5 (Man-in-the-Middle Security for Passwordless Authentication). For PlANRK $\mathsf{PlANRK} = (\mathsf{Gen}, \mathsf{Reg}, \mathsf{Auth})$, let us define the following security experiment **MiM** run between the challenger and an adversary $\mathcal{A}$.

- **Setup.** For each token $T \in \mathcal{T}$, a key is generated by running $\mathsf{msk}_T \leftarrow \mathsf{Gen}(\mathsf{par})$. Then, each token T is initialized with key msk_T and each server $S \in \mathcal{S}$ is initialized with an empty state.
- **Phase 1.** In the online phase the adversary is allowed to interact with the oracles Start, Challenge, Complete as in Definition 3.3.
- **Output Phase.** Finally, the adversary terminates the experiment which return 1 if and only if there exists a server oracle $\pi_S^{i,j}$ for $j > 0$ such that the following conditions hold:
  1. $\pi_S^{i,0}$ is partnered with a token oracle $\pi_T^{k,0}$.
  2. $\pi_S^{i,j}$ accepted, i.e. in the call $\mathsf{Complete}(\pi_S^{i,j}, \mathsf{cid}, R)$ the algorithm $\mathsf{acheck}(\mathsf{cid}, R)$ returned 1.
  3. $\pi_S^{i,j}$ is not partnered with any token oracle $\pi_T^{i',j'}$.

We define the advantage of $\mathcal{A}$ in winning the experiment as:

$$\mathbf{Adv}^{\mathcal{A}}_{\mathbf{MiM},\mathsf{PlANRK}} := \Pr[\mathbf{MiM}^{\mathcal{A}}_{\mathsf{PlANRK}} = 1].$$

3.2 Defining Unlinkability

We will now discuss the problem of unlinkability for passwordless authentication. Informally, what we want to achieve is that interactions between the same token and different servers are unlinkable, even if servers are malicious. If a protocol is proven secure using the below definition then it means the passwordless authentication protocol for non-resident keys does not provide means to malicious servers to link interactions. This also means that data that is exchanged outside of the protocol is out of the scope of our definition, e.g. metadata that could be used to link interactions of the token.

Definition 3.6 (Unlinkability for Passwordless Authentication). For PlANRK $\mathsf{PlANRK} = (\mathsf{Gen}, \mathsf{Reg}, \mathsf{Auth})$, define the security experiment **Unl** run between the challenger and an adversary $\mathcal{A}$ as follows.

- **Setup.** For each token $T \in \mathcal{T}$, a key is generated by running $\mathsf{msk}_T \leftarrow \mathsf{Gen}(\mathsf{par})$. Then, each token T is initialized with key msk_T and each server $S \in \mathcal{S}$ is initialized with an empty state.
- **Phase 1.** In the online phase the adversary is allowed to interact with the oracles Start, Challenge, Complete as in Definition 3.3.
- **Phase 2.** In this phase, the adversary outputs two user identifiers T_0, T_1, and two (not necessarily distinct) server identifier $S_L, S_R \in \mathcal{S}$. Let i_0 and i_1 be the smallest identifiers for which the token instances $\pi_{T_0}^{i_0,0}$ and $\pi_{T_1}^{i_1,0}$ were not queried to the Challenge oracle in phase 1. The challenger now chooses a bit b and initializes two oracles and keeps internally two values j_0, j_1 initialized to 0:
  - $\mathsf{Left}(\mathsf{cid}, M)$: Return $\mathsf{Challenge}(\pi_{T_b}^{i_b,j_b}, \mathsf{id}_{S_L}, \mathsf{cid}, M)$ and increase $j_b = j_b + 1$,
  - $\mathsf{Right}(\mathsf{cid}, M)$: Return $\mathsf{Challenge}(\pi_{T_{1-b}}^{i_{1-b},j_{1-b}}, \mathsf{id}_{S_R}, \mathsf{cid}, M)$ and increase $j_{1-b} = j_{1-b} + 1$
- **Phase 3.** The adversary is allowed to interact with all the oracles defined in Phase 1 and 2.
- **Output Phase.** Finally, the adversary outputs a bit $\hat{b}$ and the experiment returns 1 if and only if:
  - bit $\hat{b}$ is equal to bit b, and
  - (intance freshness) the adversary never made a query to the Challenge oracle using the token oracle instances $\pi_T^{i_0,k_0}$ and $\pi_T^{i_1,k_1}$ for any k_0, k_1, and
  - (registration uniqueness) there was never a query to the $\mathsf{Challenge}(\pi_T^{i,j}, \mathsf{id}_S, \cdot, \cdot)$ oracle for any i, j, $T \in \{T_0, T_1\}$ and $S \in \{S_L, S_R\}$.

We define the advantage of $\mathcal{A}$ in winning the experiment as:

$$\mathbf{Adv}^{\mathcal{A}}_{\mathbf{Unl},\mathsf{PlANRK}} := |\Pr[\mathbf{Unl}^{\mathcal{A}}_{\mathsf{PlANRK}} = 1] - 1/2|.$$

We will now briefly discuss the idea behind this definition and the winning condition. First notice that we reuse the oracles from the impersanation experiments. This allows the adversary to create a view on the system that shows all the connections, i.e. which account (e.g. defined by cid) corresponds to which authenticator. However, such a powerfull adverary would immediately break any unlinkability definition by trivially revealing those connection. The key point of our unlinkability definition is for the adversary to reveal connections between two tokens and two servers with the knowledge of every other connection in the system.

Therefore, we restrict the adversary and ask it to output two tokens and two servers which will be part of the challenge. Informally, we give the adversary a left and right oracle that are initialized depending on a bit b with one of the two tokens returned by the adversary. If the adversary tries to interact with those tokens using one of the two returned servers then it must do it via the left or right oracle. We ensure this by the registration uniqueness property where we only allow to register the target tokens at most once with those servers. To not allow for trivial attack we also do not allow the adversary to interact with instances of the tokens that we initialized the left and right oracles with. We call this condition instance freshness in the definition above.

3.3 Difference to Barbosa et al.'s Model

As we mentioned in the introduction, Barbosa et al. [6] model FIDO2 passwordless authentication exclusively for the use of resident keys, i.e., where all the secret keys are stored on the authentication device. Our model, on the other hand, focuses on the more practical use case, namely authentication using non-resident keys. As a consequence, our syntax and experiment for authentication are different from those in [6]. The main novelty of our model, however, is our definition of unlinkability. Below we will summarize our changes to Barbosa et al.'s model.

(1) When using resident keys, FIDO2 uses a per-server signature counter to enable cloning detection of token devices. However, the main feature of non-resident keys is that they allow for use of token devices with limited memory. Clearly, due to limited memory, a token can not keep state for every server it interacts with. On the other hand, if the token uses a naive implementation of a global state (e.g., a counter), this might violate privacy/unlinkability of the token. Hence, most authentication tokens used in practice favor privacy over cloning detection. This change is reflected in the winning condition of the adverary in the passwordless authentication experiment. [6] ensures that if a server oracle accepts then it is partnered with a *unique* token oracle with the same session identifier. In our model we do not include this condition on the number of partnered token oracles with a single server oracle. This models the case that a single server runs multiple sessions with the same token which is the case e.g. if there exists a cloned token (and it is not detected).

(2) We extract the server identifier sid from the commands exchanged between the server and token. In contrast to [6], all of our schemes require explicit access to this identifier to provide a meaningful unlinkability definition.

(3) For the same reason, we assume that the credential identifier cid is explicitly generated on the token and shared with the server during registration (rather than wrapped together with other values as done in [6]). We use this identifier to model the key handle that is bound to the user's account and provided to the token during authentication.

(4) In contrast to [6], we do not consider attestation in this paper. As already mentioned in the introduction, attestation is rarely used in practice. What is more, if designed incorrectly, attestation can lead to trivial attacks against unlinkability. We leave a formal analysis of WebAuthn with anonymous attestation (e.g. using ECDAA [1]) for future research. To account for this change in the model, the server's state is not initialized with any public key material of the tokens.

4 EXISTING SCHEMES

There are many ways to implement non-resident keys for the FIDO2 protocol. While the protocol itself remains the same, the only difference is in the way secret keys and key handles are generated. In this paper we will focus on the following two:

- **Key Derivation Function.** - the signing key is generated in a pseudorandom way using a master secret key, the server's identifier, and the key handle which acts as a random salt. This method is e.g. used by the open-source FIDO2 token SoloKey. [2]
- **Key wrapping.** - the signing keys are encrypted together with the server's identifier and the ciphertext is stored in the key handle. This method is e.g. used by Yubico in their implementation of FIDO2 tokens [32].

We will now describe the above two schemes using our syntax for passwordless authentication. Then we will prove security against man-in-the-middle attacks. Additionally, we will also show how to instantiate the used building blocks (i.e. encryption scheme and KDF) to provide privacy. To this end, we will prove the security of the above using our unlinkability definition. We will use a common description and refer to each of those schemes as respectively kwrPA and kdfPA.

Key Generation. Initialization of a token is done using the Gen algorithm that outputs a master secret key msk. For kdfPA this key is generated by executing $\mathsf{msk} \xleftarrow{\$} \{0,1\}^{\lambda}$. In the case of the key wrapping scheme, this key is the secret key for a symmetric key encryption scheme ske. Thus, we set $\mathsf{msk} \longleftarrow \mathsf{ske.Gen}(\mathsf{param})$.

Registration. The registration protocol Reg is formally presented in Figure 2. First, the server executes algorithm $\mathsf{rchallenge}(\mathsf{id}_S)$, which generates a random nonce $rs \xleftarrow{\$} \{0,1\}^{\lambda}$ and returns the challenge $c = (\mathsf{id}_S, rs)$. The challenge is then sent to the client, which executes the $\mathsf{rcommand}(\mathsf{id}_S, c)$ algorithm. This algorithm verifies that $\mathsf{id} = \mathsf{id}_S$ and returns message $M_r = H(rs)$. Afterwards, M_r and id_S is send to the token, which executes algorithm $\mathsf{rresponse}(\mathsf{msk}, \mathsf{id}_S, M_r)$. Here, the variants kwrPA and kdfPA differ slightly.

In kdfPA the token first chooses a random identifier $\mathsf{cid} \xleftarrow{\$} \{0,1\}^{\lambda}$. It then uses the key derivation function to generate the secret key $\mathsf{sk} := KDF(\mathsf{msk}, \mathsf{cid}, \mathsf{id}_S)$. Note that we assume here that the KDF outputs values in the secret key space of the signature scheme. This allows the token to compute the corresponding public key $\mathsf{pk} := \mathsf{toPK}(\mathsf{sk})$.

[2] see https://github.com/solokeys/solo

In kwrPA the keypair is computed directly using the signature scheme key generation algorithm $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{kg}(\mathsf{param})$. The secret key sk is then encrypted by the token together with id_S as $\mathsf{cid} := \mathsf{Enc}(\mathsf{msk}, (\mathsf{id}_S, \mathsf{sk}))$.

For both schemes the secret key sk is used to create the signature $\sigma \leftarrow \mathsf{sign}(\mathsf{sk}, m)$, where $m := (H(\mathsf{id}_S), \mathsf{cid}, \mathsf{pk}, h = M_r)$. Finally, the token sends a response message $R_r = (\mathsf{pk}, \sigma)$ together with cid to the client, which forwards it directly to the server.

The server verifies the token's response by running algorithm $\mathsf{rcheck}(\mathsf{cid}, R_r)$, which is as follows. The server returns $(0, \perp)$ in case $\mathsf{ver}(\mathsf{pk}, \sigma, m)$, where $m := (H(\mathsf{id}_S), \mathsf{cid}, \mathsf{pk}, M_r)$. It then updates the internal registration state with the new registration $\mathsf{rcs}[\mathsf{cid}] := \mathsf{pk}$ and outputs $(1, \mathsf{cred} := (\mathsf{cid}, \mathsf{pk}))$.

Authentication. The authentication protocol Auth, given in Figure 3, begins with the server executing the $\mathsf{achallenge}(\mathsf{id}_S)$ algorithm, which generates a random nonce $rs \xleftarrow{\$} \{0,1\}^\lambda$ and returns the challenge $c = (\mathsf{id}_S, rs)$. The challenge and the identifier cid is sent to the client. Given the challenge c from the server the client executes the $\mathsf{acommand}(\mathsf{id}_S, c)$ algorithm, which parses the challenge as $c := (\mathsf{id}, rs)$, verifies that $\mathsf{id} = \mathsf{id}_S$ and sends the message $M_a := H(rs)$, the server identifier id_S and identifier cid to the token. The token executes the $\mathsf{aresponse}(\mathsf{msk}, \mathsf{id}_S, \mathsf{cid}, M_a)$ algorithm, which is described in the following.

In the first step the token recreates the signing key sk that it created during registration. For kdfPA this means that the token runs $\mathsf{sk} := KDF(\mathsf{msk}, \mathsf{cid}, \mathsf{id}_S)$. In case of the kwrPA the token first decrypts the cid to receive $(\mathsf{id}, \mathsf{sk}') := \mathsf{Dec}(\mathsf{msk}, \mathsf{cid})$. It then checks if this secret key correspond to the server, i.e. it returns an error if $\mathsf{id} \neq \mathsf{id}_S$. Otherwise it sets $\mathsf{sk} = \mathsf{sk}'$.

For both schemes the secret key sk is used to create the signature $\sigma \leftarrow \mathsf{sign}(\mathsf{sk}, m)$, where $m := (H(\mathsf{id}_S), M_a)$. Finally, the token creates a response message $R_a = (\sigma)$, which is send to the client. The client forwards this message directly to the server.

The server verifies the token's response by running algorithm $\mathsf{acheck}(\mathsf{cid}, R_a)$, which is as follows. In the first step, the server uses the internal state to get the token's public key $\mathsf{pk} \leftarrow \mathsf{rcs}[\mathsf{cid}]$ using the identifier cid. Finally, it sets $m := (H(\mathsf{id}_S), H(rs))$ and outputs $\mathsf{ver}(\mathsf{pk}, \sigma, m)$.

Security With Key Derivation Function. We will now show man-in-the-middle security and unlinkability for the scheme kdfPA, which uses a key derivation function.

Lemma 4.1. *Let $\mathcal{A}$ be an adversary in the man-in-the-middle game of* kdfPA. *Assume that $\mathcal{A}$ makes at most Q_{KDF}, Q_H queries to random oracles KDF, H, respectively, at most Q_S queries to oracle* Start, *and at most Q_C queries to oracle* Challenge. *Then there exists an algorithm $\mathcal{B}$ with the same running time as $\mathcal{A}$ such that $\mathcal{A}$'s advantage in the man-in-the-middle game can be upper bounded by*

$$\frac{Q_{KDF} \cdot |\mathcal{T}| + |\mathcal{T}|^2 + Q_S^2}{2^\lambda} + \frac{Q_H^2}{2^{2\lambda}} + Q_C \cdot \mathbf{Adv}^{\mathcal{B}}_{\text{euf-cma,sig}}.$$

Proof. We show the statement via a sequence of games. For each game $\mathbf{G}_i$, we denote the advantage of $\mathcal{A}$, i.e. probability that $\mathbf{G}_i$ outputs 1, by $\mathbf{Adv}_i$.

Game $\mathbf{G}_0$: This is the real man-in-the-middle game. Recall that at the beginning, each token $T \in \mathcal{T}$ is initialized with a master secret key $\mathsf{msk}_T \xleftarrow{\$} \{0,1\}^\lambda$. Then, the adversary gets access to oracles Start, Challenge, Complete. By definition, we have

$$\mathbf{Adv}_0 = \mathbf{Adv}^{\mathcal{A}}_{\mathbf{MiM},\mathsf{kdfPA}}.$$

Game $\mathbf{G}_1$: This game is as $\mathbf{G}_0$, but we introduce a bad event and let the game abort if this bad event occurs. The bad event occurs, if the adversary ever queries $KDF(\mathsf{msk}_T, \cdot)$ for any token $T \in \mathcal{T}$. Note that $\mathcal{A}$ obtains no about msk_T throughout the game (except via hash values). Thus, we can use a union bound over the Q_{KDF} random oracle queries and the $|\mathcal{T}|$ tokens to obtain

$$|\mathbf{Adv}_0 - \mathbf{Adv}_1| \leq \frac{Q_{KDF} \cdot |\mathcal{T}|}{2^\lambda}.$$

Game $\mathbf{G}_2$: This game is as $\mathbf{G}_1$, but we introduce another bad event and let the game abort if this bad event occurs. The bad event occurs, if there are tokens $T \neq T'$ such that $\mathsf{msk}_T = \mathsf{msk}_{T'}$. Clearly, we have

$$|\mathbf{Adv}_1 - \mathbf{Adv}_2| \leq \frac{|\mathcal{T}|^2}{2^\lambda}.$$

Game $\mathbf{G}_3$: We rule out another bad event. That is, we let the game abort if this bad event occurs. Namely, we abort whenever there is a collision for random oracle H. Precisely, we abort, if for $x \neq x'$ we have $H(x) = H(x')$. As the images of H are sampled uniformly at random from $\{0,1\}^{2\lambda}$, we have

$$|\mathbf{Adv}_2 - \mathbf{Adv}_3| \leq \frac{Q_H^2}{2^{2\lambda}}.$$

Game $\mathbf{G}_4$: In this game, we introduce yet another bad event, for which we let the game abort on its occurrence. To define this event, consider the server-side oracles Start. Recall that these oracles (in both registration and authentication) sample a random $rs \xleftarrow{\$} \{0,1\}^{\geq\lambda}$. The bad event occurs if the same rs is sampled in two different invocations of the oracle Start. Clearly, we can bound the distinguishing advantage by

$$|\mathbf{Adv}_3 - \mathbf{Adv}_4| \leq \frac{Q_S^2}{2^\lambda}.$$

Finally, we bound the probability $\mathbf{Adv}_4$ that $\mathbf{G}_4$ outputs 1. Recall that the game outputs 1 if none of the introduced aborts occur, and the adversary successfully finished an authentication via oracle $\mathsf{Complete}(\pi_S^{i,j}, \cdot)$, for which $j = 0$, the oracle $\pi_S^{i,j}$ is not partnered with any oracle $\pi_T^{i',j'}$ and the oracle $\pi_S^{i,0}$ is partnered with an oracle $\pi_T^{i',0}$. In the following, we will call this interaction with oracle Complete the *forged authentication.* We refer to the interaction with oracle $\pi_T^{i',0}$ as above via oracle Challenge as the *target registration.*

Now, we give a reduction $\mathcal{B}$ from the euf-cma security of sig. The reduction is as follows.

- $\mathcal{B}$ gets as input a public key pk^*. It gets access to a signing oracle Sign.
- $\mathcal{B}$ samples an index $k^* \xleftarrow{\$} [Q_C]$. It simulates game $\mathbf{G}_4$, except for the k^*-th query to oracle Challenge, for which it works as follows:
  - If this query is an authentication query, i.e. it is of the form $\mathsf{Challenge}(\pi_T^{i,j}, \cdot)$ for $j > 0$, then it the reduction $\mathcal{B}$ aborts.

  - If this query is a registration query, i.e. it is of the form $\mathsf{Challenge}(\pi_T^{i,0}, \cdot)$, then it sets $\mathsf{pk} := \mathsf{pk}^*$, and continues the simulation using this key. Thereby, it obtains the signature σ from its signing oracle Sign. Let cid be the random credential identifier sampled within this simulation. Later, whenever the adversary queries oracle Challenge for authentications for this cid, the reduction uses its signing oracle Sign to answer the query.
- After termination of $\mathcal{A}$, reduction $\mathcal{B}$ first finds the forged authentication and the corresponding target registration.
- If the target registration is not the k^*-th query to oracle Challenge, $\mathcal{B}$ aborts. Otherwise, let σ be the signature that $\mathcal{A}$ submitted in the forged authentication, id_S be the server identity that is used in the forged authentication, and rs be the challenge that is used.
- $\mathcal{B}$ returns

$$m^* := (H(\mathsf{id}_S), H(rs)), \sigma^* := \sigma.$$

First, assume that $\mathcal{B}$ does not abort. It is easy to see that, the simulation of game $\mathbf{G}_4$ is perfect. Also, by definition of algorithm acheck, as the forged authentication accepted, σ^* is a valid signature on m^* under pk^*. Furthermore, due to the changes introduced in games $\mathbf{G}_3$ and $\mathbf{G}_4$, if $\mathcal{A}$ wins game $\mathbf{G}_4$, we know that the signing oracle Sign was never used on a message $(\cdot, H(rs))$, and thus the forgery output by $\mathcal{B}$ is fresh. Finally, note that $\mathcal{A}$'s view is independent of k^*, until an abort happens. Thus, we have

$$\mathbf{Adv}_4 \leq Q_C \cdot \mathbf{Adv}^{\mathcal{B}}_{\mathsf{euf\text{-}cma,sig}}.$$

□

Lemma 4.2. *Let $\mathcal{A}$ be an adversary in the unlinkability game of* kdfPA. *Assume that $\mathcal{A}$ makes at most Q_{KDF} queries to random oracle KDF. Then $\mathcal{A}$'s advantage in the unlinkability game can be upper bounded by*

$$\frac{Q_{KDF} \cdot |\mathcal{T}| + |\mathcal{T}|^2}{2^\lambda}.$$

Proof. We show the statement via a sequence of games. For each game $\mathbf{G}_i$, we denote the probability that $\mathbf{G}_i$ outputs 1 by pr_i. We note that games $\mathbf{G}_1, \mathbf{G}_2$ are taken verbatim from the proof of Lemma 4.1.

Game $\mathbf{G}_0$: This game is the real unlinkability game. Recall that at the beginning of this game, a master secret key $\mathsf{msk}_T \xleftarrow{\$} \{0,1\}^\lambda$ is generated for each token $T \in \mathcal{T}$. The adversary $\mathcal{A}$ gets access to oracles Start, Challenge, Complete. Then, it outputs two tokens T_0, T_1 and servers S_L, S_R. Afterwards, it also gets access to oracles Left, Right, which internally run $\mathsf{Challenge}(\pi_{T_b}^{i_b,j_b}, \mathsf{id}_{S_L}, \cdot, \cdot)$ and $\mathsf{Challenge}(\pi_{T_{1-b}}^{i_{1-b},j_{1-b}}, \mathsf{id}_{S_R}, \cdot, \cdot)$, respectively. Throughout the game, the adversary $\mathcal{A}$ also gets access to random oracle KDF. By definition, we have

$$\mathsf{pr}_0 = \mathbf{Adv}^{\mathcal{A}}_{\mathbf{Unl},\mathsf{kdfPA}}.$$

Game $\mathbf{G}_1$: This game is as $\mathbf{G}_0$, but we introduce a bad event and let the game abort if this bad event occurs. The bad event occurs, if the adversary ever queries $KDF(\mathsf{msk}_T, \cdot)$ for any token $T \in \mathcal{T}$. Note that $\mathcal{A}$ obtains no information about msk_T throughout the game (except via hash values). Thus, we can use a union bound over the Q_{KDF} random oracle queries and the $|\mathcal{T}|$ tokens to obtain

$$|\mathsf{pr}_0 - \mathsf{pr}_1| \leq \frac{Q_{KDF} \cdot |\mathcal{T}|}{2^\lambda}.$$

Game $\mathbf{G}_2$: This game is as $\mathbf{G}_1$, but we introduce another bad event and let the game abort if this bad event occurs. The bad event occurs, if there are tokens $T \neq T'$ such that $\mathsf{msk}_T = \mathsf{msk}_{T'}$. Clearly, we have

$$|\mathsf{pr}_1 - \mathsf{pr}_2| \leq \frac{|\mathcal{T}|^2}{2^\lambda}.$$

Game $\mathbf{G}_3$: This game is as $\mathbf{G}_2$, but we change how oracle Left behaves. Recall that in $\mathbf{G}_2$, oracle Left runs $\mathsf{Challenge}(\pi_{T_b}^{i_b,j_b}, \mathsf{id}_{S_L}, \cdot, \cdot)$. Concretely, in each call $\mathsf{Left}(\mathsf{cid}, M)$ in $\mathbf{G}_2$, a key pair $(\mathsf{sk}, \mathsf{pk})$ is sampled as

$$\mathsf{sk} := KDF(\mathsf{msk}_{T_b}, \mathsf{cid}, \mathsf{id}_{S_L}),\ \mathsf{pk} := \mathsf{toPK}(\mathsf{sk}).$$

In game $\mathbf{G}_3$ we change the way these keys are generated. Namely, we instead keep a map $K_L[\cdot]$, and in each call $\mathsf{Left}(\mathsf{cid}, M)$ we first check if $K_L[\mathsf{cid}]$ is already defined. If it is defined, we use it as a key pair. Otherwise, we sample a fresh key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{kg}(\mathsf{param})$, store it as $K_L[\mathsf{cid}] := (\mathsf{sk}, \mathsf{pk})$ and use it. The rest of the oracle stays the same, using that particular key pair.

We claim that $\mathsf{pr}_2 = \mathsf{pr}_3$. This can be argued as follows. Without loss of generality, we can focus on the case where $\mathcal{A}$ does not violate registration uniqueness or instance freshness. Also, we can assume that the bad events defined in games $\mathbf{G}_1, \mathbf{G}_2$ do not occur. In case any of these assumptions does not hold, both games $\mathbf{G}_2$ and $\mathbf{G}_3$ output 0. Using these assumptions, we see that the adversary learns nothing about $KDF(\mathsf{msk}_{T_b}, \mathsf{cid}, \mathsf{id}_{S_L})$ in game $\mathbf{G}_1$. Especially, due to registration uniqueness and non-colliding msk's, the oracle Challenge never queries $KDF(\mathsf{msk}_{T_b}, \mathsf{cid}, \mathsf{id}_{S_L})$. Thus, $KDF(\mathsf{msk}_{T_b}, \mathsf{cid}, \mathsf{id}_{S_L})$ is uniformly random for $\mathcal{A}$, which implies that the distribution of key pairs used is exactly the same as in game $\mathbf{G}_3$. We have

$$\mathsf{pr}_2 = \mathsf{pr}_3.$$

Game $\mathbf{G}_4$: This game is as $\mathbf{G}_3$, but we change how oracle Right behaves. Concretely, we apply a similar change using fresh key pairs and a map K_R as we did for oracle Left in game $\mathbf{G}_3$. With the same arguments, it follows that

$$\mathsf{pr}_3 = \mathsf{pr}_4.$$

Finally, we see that the behavior of oracles Left and Right in game $\mathbf{G}_4$ and thus $\mathcal{A}$'s view is independent of the bit b. Thus, we have $\mathsf{pr}_4 = 1/2$. This shows the claim. □

Security of Key Wraping. We will now show man-in-the-middle security and unlinkability for the scheme kwrPA, which uses key wrapping. Notably, our proofs require that the encryption scheme that is used is anonymous. To the best of our knowledge, it is not known if the CCM mode used in Yubico's implementation of key wrapping is anonymous. Due to space limitations, we postpone the proofs of the statements to Appendix A.

Lemma 4.3. *Let $\mathcal{A}$ be an adversary in the man-in-the-middle game of* kwrPA. *Assume that $\mathcal{A}$ makes at most Q_H queries to random oracle H, at most Q_S queries to oracle* Start, *and at most Q_C queries to oracle* Challenge. *Then there exist algorithms $\mathcal{B}$ and C with the same* *running time as $\mathcal{A}$ such that $\mathcal{A}$'s advantage in the man-in-the-middle game can be upper bounded by*

$$\frac{Q_S^2}{2^\lambda} + \frac{Q_H^2}{2^{2\lambda}} + |\mathcal{T}| \cdot \mathbf{Adv}^{\mathcal{B}}_{\textit{anon-auth},\mathsf{ske}} + Q_C \cdot \mathbf{Adv}^{C}_{\text{euf-cma},\mathsf{sig}}.$$

Lemma 4.4. *Let $\mathcal{A}$ be an adversary in the unlinkability game of* kwrPA. *Then there exists an algorithm $\mathcal{B}$, which has the same running time as $\mathcal{A}$, such that $\mathcal{A}$'s advantage in the unlinkability game can be upper bounded by*

$$|\mathcal{T}^2| \cdot \mathbf{Adv}^{\mathcal{B}}_{\textit{anon-auth},\mathsf{ske}}.$$

5 GLOBAL KEY REVOCATION

A useful feature that FIDO2 currently does not support is a means of revoking keys of a comprised token. Informally, we want to give the user the option to revoke its keys globally, without accessing all the servers which her token is registered to one by one. In this section, we focus on formally defining syntax and security for global key revocation. Then, in the next section we show a way to add this feature to FIDO2 using BIP32 key derivation.

In short, we define a global revocation mechanism as two algorithms that are associated with PlANRK. Intuitively, these should be understood as follows. First, when a user starts using its token $T \in \mathcal{T}$, it also obtains a revocation key rk, which is extracted from the long-term key msk using some algorithm Revoke. Recall from Definition 3.1 that when token T registers at a server $S \in \mathcal{S}$, the server stores a credential cred for this token in its state. If the user wants to revoke its key, it publishes rk. We do not further specify how the user publishes rk. However, we assume that all servers periodically scan for published revocation keys rk. Whenever a new revocation key is published, the server (with identity id_S) runs an algorithm $\mathsf{CheckCred}(\mathsf{id}_S, \mathsf{cred}, \mathsf{rk})$ for each credential cred in its state. If this algorithm accepts, the server considers this credential as revoked.

Definition 5.1 (Global Key Revocation of PlANRK). A PlANRK PlANRK = (Gen, Reg, Auth) satisfies global key revocation if there are algorithms Revoke, CheckCred with the following syntax:

- Revoke(msk) takes as input a master secret key msk and outputs a revocation key rk.
- $\mathsf{CheckCred}(\mathsf{id}_S, \mathsf{cred}, \mathsf{rk})$ takes as input a server identity id_S, a credential cred and a revocation key rk and outputs a bit $b \in \{0, 1\}$.

Further, the algorithms should be complete in the following sense: For all msk ∈ Gen(par), parties T and S, states St_T, St_S the following experiment outputs 1 with probability 1:

(1) Run steps (1)-(3) from the experiment in Definition 3.2.
(2) If $b_a = 0$, output 0. Otherwise, run $\mathsf{rk} \leftarrow \mathsf{Revoke}(\mathsf{msk})$ and $b \leftarrow \mathsf{CheckCred}(\mathsf{id}_S, \mathsf{cred}, \mathsf{rk})$. Return b.

Clearly, the above definition is easily satisfied if we make algorithm CheckCred always output $b = 1$. This is not what we aim for. Instead, we need a security notion that ensures that only the owner of msk can revoke keys that are stored on an honest server.

Definition 5.2 (Revocation Soundness of PlANRK). Let PlANRK = (Gen, Reg, Auth) be a PlANRK. Consider an algorithm $\mathcal{A}$ and the following experiment $\mathbf{rev\text{-}sound}^{\mathcal{A}}_{\mathsf{PlANRK}}$:

- **Setup.** For each token $T \in \mathcal{T}$, a key is generated by running $\mathsf{msk}_T \leftarrow \mathsf{Gen}(\mathsf{par})$. Then, each token T is initialized with key msk_T and each server $S \in \mathcal{S}$ is initialized with an empty state.
- **Online Phase 1.** $\mathcal{A}$ gets as input $\{\mathsf{mpk}_T\}_{T \in \mathcal{T}}$ and oracle access to oracles Start, Challenge, Complete as in Definition 3.3. Then, $\mathcal{A}$ outputs a token T^* and a server S^*.
- **Registration.** Run the registration protocol Reg of T^* at S^*, which is as follows:

$$c \leftarrow \mathsf{rchallenge}(\mathsf{id}_{S^*}),$$
$$M_r \leftarrow \mathsf{rcommand}(\mathsf{id}_{S^*}, c),\ (\mathsf{cid}, R_r) \leftarrow \mathsf{rresponse}(\mathsf{msk}_{T^*}, \mathsf{id}_{S^*}, M_r),$$
$$(b_r, \mathsf{cred}) \leftarrow \mathsf{rcheck}(\mathsf{cid}, R_r).$$

- **Online Phase 2.** Continue the execution of $\mathcal{A}$ with the same oracles as before. Additionally, provide the input $(c, M_r, \mathsf{cid}, R_r, b_r, \mathsf{cred})$ to $\mathcal{A}$.
- **Challenge Phase.** The adversary outputs rk^*. The experiment outputs 1 if and only if $\mathsf{CheckCred}(\mathsf{id}_{S^*}, \mathsf{cred}, \mathsf{rk}^*) = 1$.

We define the advantage of $\mathcal{A}$ in $\mathbf{rev\text{-}sound}^{\mathcal{A}}_{\mathsf{PlANRK}}$ as

$$\mathbf{Adv}^{\mathcal{A}}_{\mathbf{rev\text{-}sound},\mathsf{PlANRK}} := \Pr[\mathbf{rev\text{-}sound}^{\mathcal{A}}_{\mathsf{PlANRK}} = 1].$$

To show our informal claim that even for globally revoked keys impersonation is impossible, we extend the notion of impersonation security.

Definition 5.3 (Man-in-the-Middle Security with Global Revocation). Let PlANRK = (Gen, Reg, Auth) be a PlANRK. We consider the experiment given in Definition 3.5 with the following modification:

After generating msk_T for each $T \in \mathcal{T}$, the experiment also generates $\mathsf{rk}_T \leftarrow \mathsf{Revoke}(\mathsf{msk}_T)$ for each $T \in \mathcal{T}$. Then, $\{\mathsf{rk}_T\}_{T \in \mathcal{T}}$ is given to algorithm $\mathcal{A}$ as an additional input.

We define the advantage of an algorithm $\mathcal{A}$ in the experiment as

$$\mathbf{Adv}^{\mathcal{A}}_{\mathbf{MitM\text{-}GR},\mathsf{PlANRK}} := \Pr[\mathbf{MitM\text{-}GR}^{\mathcal{A}}_{\mathsf{PlANRK}} = 1].$$

6 BIP32 PASSWORDLESS AUTHENTICATION

In this section, we show how to instantiate FIDO2 by deriving ECDSA keypairs using the BIP32 key derivation [31] The resulting scheme, denoted by bip32PA supports global key revocation.

Key-Prefixed ECDSA.. We recall the necessary background of the ECDSA signature scheme. We denote the scheme by sig = (kg, sign, srerand, prerand, ver). For the purpose of this work, we can treat signing and verification as a black box. For details, see [14]. We highlight that we use a variant of ECDSA, where messages are prefixed with the public key, see [14]. We now describe key generation and randomization. The system parameters of the scheme contain a group $\mathbb{G}$ of prime order p with generator $g \in \mathbb{G}$. Key generation and rerandomization, i.e. algorithms kg, srerand, prerand, are as follows:

- Secret keys are sampled uniformly from $\mathbb{Z}_p$, i.e. $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_p$.
- The public key for a secret key sk is $\mathsf{pk} := g^{\mathsf{sk}}$.

- To rerandomize a secret key with randomness $\rho \in \mathbb{Z}_p$, we compute $\mathsf{pk}' := \mathsf{pk} \cdot g^{\rho}$ and $\mathsf{sk}' := \mathsf{sk} + \rho$.

In terms of security, Das et al. [14] show that the key-prefixed ECDSA scheme is **ufcma-hrk**1. We will use this result to show the security of our construction.

Lemma 6.1 (informal, [14]). *Let* sig *be the key-prefixed ECDSA signature scheme as above. Then, under a suitable assumption, for each efficient algorithm $\mathcal{A}$, the advantage $\mathbf{Adv}^{\mathcal{A}}_{\textbf{\textit{ufcma-hrk}}1,\mathsf{sig}}$ is negligible.*

Key Generation. Let us describe how master secret keys msk are generated for our scheme bip32PA. A master secret key msk consists of an ECDSA key pair $(\mathsf{sk}_0, \mathsf{pk}_0)$ and a so called chaincode ch. Looking ahead, the chaincode and the public key pk_0 can later be used to revoke keys. Concretely, the components of the key are generated as $\mathsf{sk}_0 \xleftarrow{\$} \mathbb{Z}_p, \mathsf{pk}_0 := g^{\mathsf{sk}_0}, ch \xleftarrow{\$} \{0,1\}^{\lambda}$.

Registration and Authentication. Registration and authentication follow the FIDO2 specification. Thus, they are very similar to the protocols described in Section 4. Formally, we present protocols Reg and Auth in Figures 2 and 3. Let us describe the differences between bip32PA and the existing schemes. The most important difference is how keys are derived during registration and authentication. Namely, the token chooses a random $\mathsf{cid} \xleftarrow{\$} \{0,1\}^{\lambda}$ as in the scheme kdfPA. Then, it derives a randomness $\rho := \hat{H}(\mathsf{pk}_0, ch, \mathsf{cid}, \mathsf{id}_S)$ using a random oracle $\hat{H}$. This randomness is used to rerandomize the pair $(\mathsf{sk}_0, \mathsf{pk}_0)$ to a new keypair, i.e. $\mathsf{sk} := \mathsf{srerand}(\mathsf{sk}_0, \rho)$ and $\mathsf{pk} := \mathsf{prerand}(\mathsf{pk}_0, \rho)$. As in the scheme kdfPA, the server obtains and stores $\mathsf{cid}, \mathsf{pk}$ during registration and the key sk is used to sign challenges. One minor detail that we have to change is the use of an additional nonce $no \xleftarrow{\$} \{0,1\}^{\lambda}$ that is sampled by the token and added to the actual challenge before signing. This ensures that with high probability, even for malicious servers a token never signs the same message twice. We need this, as the security notion **ufcma-hrk**1 that ECDSA satisfies only allows one signature query per message.

Global Revocation. The advantage of the BIP32 key derivation compared to key wrapping or simple key derivation is global key revocation, as defined in the previous section. Informally, the scheme bip32PA allows to publish a revocation key rk to globally revoke a lost/stolen token without compromising the user's account. Formally, we present algorithms Revoke and CheckCred for our scheme bip32PA. Algorithm Revoke(msk) is given as follows:

(1) Parse $(\mathsf{sk}_0, \mathsf{pk}_0, ch) := \mathsf{msk}$.
(2) Return $\mathsf{rk} := (\mathsf{pk}_0, ch)$.

Algorithm $\mathsf{CheckCred}(\mathsf{id}_S, \mathsf{cred}, \mathsf{rk})$ is as follows:

(1) Parse $(\mathsf{pk}_0, ch) := \mathsf{rk}$ and $(\mathsf{cid}, \mathsf{pk}) := \mathsf{cred}$.
(2) Run $\mathsf{pk}' := \mathsf{prerand}(\mathsf{pk}_0, \hat{H}(\mathsf{pk}_0, ch, \mathsf{cid}, \mathsf{id}_S))$.
(3) Return 1 if $\mathsf{pk} = \mathsf{pk}'$. Otherwise, return 0.

6.1 BIP32 Scheme Security

Next, we show security of our scheme. Concretely, we first show that only the owner of a token can revoke its keys (i.e. revocation soundness). Then, we show unlinkability. Finally, we show that even given all revocation keys, man-in-the-middle security still holds. Due to space limitations, we postpone the proofs of revocation soundness and unlinkability to Appendix B.

Lemma 6.2. *Let $\mathcal{A}$ be an adversary in the revocation soundness game of* bip32PA. *Assume that $\mathcal{A}$ makes at most $Q_{\hat{H}}$ queries to oracle $\hat{H}$. Then the advantage of $\mathcal{A}$ against revocation soundness can be upper bounded by*

$$\mathbf{Adv}^{\mathcal{A}}_{\textbf{\textit{rev-sound}},\mathsf{PIANRK}} \leq \frac{3Q_{\hat{H}}+1}{2^{\lambda}}.$$

Lemma 6.3. *Let $\mathcal{A}$ be an adversary in the unlinkability game of* bip32PA. *Assume that $\mathcal{A}$ makes at most $Q_{\hat{H}}$ queries to random oracle $\hat{H}$. Then $\mathcal{A}$'s advantage in the unlinkability game can be upper bounded by*

$$\frac{Q_{\hat{H}} \cdot |\mathcal{T}| + |\mathcal{T}|^2}{2^{\lambda}}.$$

Lemma 6.4. *Let $\mathcal{A}$ be an adversary in the man-in-the-middle with global revocation game with global revocation of* bip32PA. *Assume that $\mathcal{A}$ makes at most $Q_H, Q_{\hat{H}}$ queries to random oracles $H, \hat{H}$, respectively, at most Q_S queries to oracle* Start, *and at most Q_C queries to oracle* Challenge. *Then there exists an algorithm $\mathcal{B}$ with the same running time as $\mathcal{A}$ such that $\mathcal{A}$'s advantage in the man-in-the-middle with global revocation game can be upper bounded by*

$$\frac{Q_S^2 + Q_C^2 + Q_C \cdot Q_{\hat{H}}}{2^{\lambda}} + \frac{Q_H^2}{2^{2\lambda}} + \mathbf{Adv}^{\mathcal{B}}_{\textbf{\textit{ufcma-hrk}}1,\mathsf{sig}}.$$

Proof. We show the statement by presenting a sequence of games $\mathbf{G}_i$, where for each such game $\mathbf{G}_i$ the probability that the game outputs 1 is denoted by $\mathbf{Adv}_i$.

Game $\mathbf{G}_0$: This game is the real man-in-the-middle with global revocation game. That is, at the beginning a master secret key $\mathsf{msk}_T = (\mathsf{sk}_{T,0}, \mathsf{pk}_{T,0}, ch_T)$ is generated for each token $T \in \mathcal{T}$. Further, the global revocation keys $\mathsf{rk}_T := (\mathsf{pk}_{T,0}, ch_T)$ are passed to the adversary. Then, the adversary $\mathcal{A}$ gets access to oracles Start, Challenge, Complete. By definition, we have

$$\mathbf{Adv}_0 = \mathbf{Adv}^{\mathcal{A}}_{\mathbf{MitM\text{-}GR},\mathsf{bip32PA}}.$$

Game $\mathbf{G}_1$: This game is as $\mathbf{G}_0$, but we introduce an additional abort. Namely, $\mathbf{G}_1$ aborts whenever for we have $H(x) = H(x')$ for $x \neq x'$. As the images of H are sampled uniformly at random from $\{0,1\}^{2\lambda}$, we have

$$|\mathbf{Adv}_0 - \mathbf{Adv}_1| \leq \frac{Q_H^2}{2^{2\lambda}}.$$

Game $\mathbf{G}_2$: In this game, another abort is introduced. Namely, we consider the server-side oracles Start. Recall that in an execution of such an oracle, the game samples a random value $rs \xleftarrow{\$} \{0,1\}^{\geq\lambda}$. Now, game $\mathbf{G}_2$ aborts if the same rs is sampled in two different invocations of the oracle Start. Clearly, we have

$$|\mathbf{Adv}_1 - \mathbf{Adv}_2| \leq \frac{Q_S^2}{2^{\lambda}}.$$

Game $\mathbf{G}_3$: In this game, we introduce another abort. Consider the token-side oracle Challenge. More precisely, consider a query $\mathsf{Challenge}(\pi_T^{i,j}, \mathsf{id}_S, \mathsf{cid}, M)$ with $j > 0$, i.e. an authentication interaction. Recall that in the execution of this oracle, a value $no \xleftarrow{\$} \{0,1\}^{\lambda}$ is sampled, and then the message $m := (M, no)$ is signed.

The game now aborts, if this message has been signed in any call to oracle Challenge before. As *no* is sampled uniformly at random and there are at most Q_C such queries, we have

$$|\mathbf{Adv}_2 - \mathbf{Adv}_3| \leq \frac{Q_C^2}{2^\lambda}.$$

Game $\mathbf{G}_4$: In this game, we change how the master secret keys $\mathsf{msk}_T = (\mathsf{sk}_{T,0}, \mathsf{pk}_{T,0}, ch_T)$ for all tokens $T \in \mathcal{T}$ are generated at the beginning of the game. In previous games, $(\mathsf{sk}_{T,0}, \mathsf{pk}_{T,0})$ was generated as a fresh key pair $(\mathsf{sk}_{T,0}, \mathsf{pk}_{T,0}) \leftarrow \mathsf{kg}(\mathsf{param})$. In game $\mathbf{G}_4$, a key pair $(\mathsf{sk}^*, \mathsf{pk}^*) \leftarrow \mathsf{kg}(\mathsf{param})$ is generated first. Then, the key pairs $(\mathsf{sk}_{T,0}, \mathsf{pk}_{T,0})$ are generated using

$$(\mathsf{sk}_{T,0}, \mathsf{pk}_{T,0}) := (\mathsf{srerand}(\mathsf{sk}^*, \rho_T), \mathsf{prerand}(\mathsf{pk}^*, \rho_T))$$

for $\rho_T \overset{\$}{\leftarrow} \{0,1\}^\lambda$. The generation of ch_T stays as in game $\mathbf{G}_4$. As the distribution of rerandomized keys is exactly the same as the distribution of fresh keys, we have

$$\mathbf{Adv}_3 = \mathbf{Adv}_4.$$

Game $\mathbf{G}_5$: In this game, we introduce another abort. Namely, consider a query of the form $\mathsf{Challenge}(\pi_T^{i,0}, \mathsf{id}_S, M)$, i.e. a token-side registration query. Recall that in the execution of such a query, the game samples $\mathsf{cid} \overset{\$}{\leftarrow} \{0,1\}^\lambda$. Later, keys $(\mathsf{sk}, \mathsf{pk})$ are derived as rerandomizations of $(\mathsf{sk}_{T,0}, \mathsf{pk}_{T,0})$ using the randomness $\rho := \hat{H}(\mathsf{pk}_{T,0}, ch_T, \mathsf{cid}, \mathsf{id}_S)$. Now, game $\mathbf{G}_5$ aborts, if at this point the hash value $\hat{H}(\mathsf{pk}_{T,0}, ch_T, \mathsf{cid}, \mathsf{id}_S)$ is already defined. As cid is sampled uniformly at random over $\{0,1\}^\lambda$, a union bound over the at most Q_C queries of this form and the at most $Q_{\hat{H}}$ random oracle queries leads to

$$|\mathbf{Adv}_4 - \mathbf{Adv}_5| \leq \frac{Q_C \cdot Q_{\hat{H}}}{2^\lambda}.$$

Game $\mathbf{G}_6$: In this game, we change the execution of queries of the form $\mathsf{Challenge}(\pi_T^{i,0}, \mathsf{id}_S, M)$ again. Concretely, the game samples $\mathsf{cid} \overset{\$}{\leftarrow} \{0,1\}^\lambda$ as before. Then, it samples a randomness $\rho \overset{\$}{\leftarrow} \{0,1\}^\lambda$ and computes $(\mathsf{sk}, \mathsf{pk})$ as rerandomizations of $(\mathsf{sk}^*, \mathsf{pk}^*)$ instead of $(\mathsf{sk}_{T,0}, \mathsf{pk}_{T,0})$. To be precise, it sets

$$(\mathsf{sk}, \mathsf{pk}) := (\mathsf{srerand}(\mathsf{sk}^*, \rho), \mathsf{pk} := \mathsf{prerand}(\mathsf{pk}^*, \rho)$$

and programs

$$\hat{H}(\mathsf{pk}_{T,0}, ch_T, \mathsf{cid}, \mathsf{id}_S) := \rho - \rho_T,$$

which is possible due to the previous change. Here the subtraction should be understood over $\mathbb{Z}_p$. It follows from the definition of ECDSA key generation that this does not change the view of the adversary. Hence we have

$$\mathbf{Adv}_5 = \mathbf{Adv}_6.$$

We want to bound $\mathbf{Adv}_6$ using a reduction from the **ufcma-hrk**1 security of sig. This is possible, as all keys involved are rerandomizations of pk^* and signing keys are only needed for signing. Further, the winning condition and the aborts that we introduced imply that a successful adversary forges a signature for a fresh message.

Before we describe the reduction, we introduce some terminology. We recall that the game outputs 1 if none of the introduced aborts occur, and the adversary successfully finished an authentication via oracle $\mathsf{Complete}(\pi_S^{i,j}, \cdot)$, for which $j = 0$, the oracle $\pi_S^{i,j}$ is not partnered with any oracle $\pi_T^{i',j'}$ and the oracle $\pi_S^{i,0}$ is partnered with an oracle $\pi_T^{i',0}$. We call this interaction with oracle Complete the *forged authentication* and the interaction with oracle $\pi_T^{i',0}$ as above via oracle Challenge the *target registration*.

Reduction $\mathcal{B}$ is as follows.

- $\mathcal{B}$ gets as input a public key pk^*. It gets access to a signing oracle Sign.
- For each token $T \in \mathcal{T}$, reduction $\mathcal{B}$ rerandomizes pk^* into $\mathsf{pk}_{T,0}$ using its oracle Rand. Thereby it also obtains the randomness ρ_T. It also samples ch_T and defines the revocation key rk_T. Then, it passes all revocation keys $\mathsf{rk}_T, T \in \mathcal{T}$ to $\mathcal{A}$.
- $\mathcal{B}$ simulates oracles Start, Challenge, Complete as in game $\mathbf{G}_6$, where it uses its oracles Sign and Rand to simulate oracle Challenge. Concretely,
  - In queries of the form $\mathsf{Challenge}(\pi_T^{i,0}, \mathsf{id}_S, M)$, $\mathcal{B}$ does not sample the randomness ρ by itself, but rather obtains ρ from its oracle Rand.
  - Whenever $\mathcal{B}$ is needs a signature according to game $\mathbf{G}_6$, it uses oracle Sign. The change introduced in game $\mathbf{G}_3$ ensures that no message is signed twice.
- After termination of $\mathcal{A}$, reduction $\mathcal{B}$ first finds the forged authentication and the corresponding target registration.
- Let σ be the signature that $\mathcal{A}$ submitted in the forged authentication, id_S be the server identity that is used in the forged authentication, and *rs* be the challenge that is used. Further, let ρ be the randomness that $\mathcal{B}$ obtained from oracle Rand in the target registration.
- $\mathcal{B}$ returns

$$\sigma^* := \sigma, \rho^* := \rho, m^* := (H(\mathsf{id}_S), H(rs)).$$

One can easily see that $\mathcal{B}$ simulates $\mathbf{G}_6$ perfectly. Further, due to the changes introduced in games $\mathbf{G}_1$ and $\mathbf{G}_2$, if $\mathcal{A}$ wins game $\mathbf{G}_6$, we know that the forgery output by $\mathcal{B}$ is fresh. We conclude with

$$\mathbf{Adv}_6 \leq \mathbf{Adv}^{\mathcal{B}}_{\mathbf{ufcma\text{-}hrk}1,\mathsf{sig}}.$$

□

7 CONCLUSIONS

In this paper we analyzed the FIDO2 protocol with a focus on its real-world use cases and adapted the existing security model of Barbosa et al. [6] accordingly. We showed that privacy (unlinkability) of the protocol is not immediately guaranteed by the specification if non-resident keys are used. To solve this issue we introduced the first formal security definition to capture privacy. Our results can be used as a guideline for token vendors. As an important example, we observed that in the case of key-wrapping the underlying encryption scheme must provide an anonymity property, i.e. ciphertexts created using the same key must be unlinkable to each other. This paper also introduces the notion of global key revocation and gives the first formalization of this property. Finally, we have shown that using the BIP32 key derivation, it is possible to obtain an efficient token implementation that supports global key revocation.

REFERENCES

[1] 2017. FIDO ECDAA Algorithm. https://fidoalliance.org/specs/fido-uaf-v1.1-ps-20170202/fido-ecdaa-algorithm-v1.1-ps-20170202.pdf. (2017). [Online; accessed 22-January-2021].

[2] 2019. Client to Authenticator Protocol (CTAP). https://fidoalliance.org/specs/fido-v2.0-ps-20190130/fido-client-to-authenticator-protocol-v2.0-ps-20190130.pdf. (2019). [Online; accessed 14-January-2022].

[3] 2020. Web Authentication: An API for accessing Public Key Credentials. https://www.w3.org/TR/webauthn/. (2020). [Online; accessed 14-January-2022].

[4] Aftab Alam, Katharina Krombholz, and Sven Bugiel. 2019. Poster: Let History not Repeat Itself (this Time) - Tackling WebAuthn Developer Issues Early On. In *ACM CCS 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM Press, 2669–2671. https://doi.org/10.1145/3319535.3363283

[5] Fabio Banfi and Ueli Maurer. 2020. Anonymous Symmetric-Key Communication. In *SCN 20 (LNCS)*, Clemente Galdi and Vladimir Kolesnikov (Eds.), Vol. 12238. Springer, Heidelberg, 471–491. https://doi.org/10.1007/978-3-030-57990-6_23

[6] Manuel Barbosa, Alexandra Boldyreva, Shan Chen, and Bogdan Warinschi. 2021. Provable Security Analysis of FIDO2. In *CRYPTO 2021, Part III (LNCS)*, Tal Malkin and Chris Peikert (Eds.), Vol. 12827. Springer, Heidelberg, Virtual Event, 125–156. https://doi.org/10.1007/978-3-030-84252-9_5

[7] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. 2003. Foundations of Group Signatures: Formal Definitions, Simplified Requirements, and a Construction Based on General Assumptions. In *EUROCRYPT 2003 (LNCS)*, Eli Biham (Ed.), Vol. 2656. Springer, Heidelberg, 614–629. https://doi.org/10.1007/3-540-39200-9_38

[8] Mihir Bellare, David Pointcheval, and Phillip Rogaway. 2000. Authenticated Key Exchange Secure against Dictionary Attacks. In *EUROCRYPT 2000 (LNCS)*, Bart Preneel (Ed.), Vol. 1807. Springer, Heidelberg, 139–155. https://doi.org/10.1007/3-540-45539-6_11

[9] Jacqueline Brendel and Marc Fischlin. 2017. Zero Round-Trip Time for the Extended Access Control Protocol. In *ESORICS 2017, Part I (LNCS)*, Simon N. Foley, Dieter Gollmann, and Einar Snekkenes (Eds.), Vol. 10492. Springer, Heidelberg, 297–314. https://doi.org/10.1007/978-3-319-66402-6_18

[10] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. 2006. How to win the clonewars: Efficient periodic n-times anonymous authentication. In *ACM CCS 2006*, Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati (Eds.). ACM Press, 201–210. https://doi.org/10.1145/1180405.1180431

[11] Dhiman Chakraborty and Sven Bugiel. 2019. simFIDO: FIDO2 User Authentication with simTPM. In *ACM CCS 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM Press, 2569–2571. https://doi.org/10.1145/3319535.3363258

[12] David Chaum and Eugène van Heyst. 1991. Group Signatures. In *EUROCRYPT'91 (LNCS)*, Donald W. Davies (Ed.), Vol. 547. Springer, Heidelberg, 257–265. https://doi.org/10.1007/3-540-46416-6_22

[13] Özgür Dagdelen and Marc Fischlin. 2011. Security Analysis of the Extended Access Control Protocol for Machine Readable Travel Documents. In *ISC 2010 (LNCS)*, Mike Burmester, Gene Tsudik, Spyros S. Magliveras, and Ivana Ilic (Eds.), Vol. 6531. Springer, Heidelberg, 54–68.

[14] Poulami Das, Andreas Erwig, Sebastian Faust, Julian Loss, and Siavash Riahi. 2021. The Exact Security of BIP32 Wallets. In *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi (Eds.). ACM, 1020–1042. https://doi.org/10.1145/3460120.3484807

[15] Poulami Das, Sebastian Faust, and Julian Loss. 2019. A Formal Treatment of Deterministic Wallets. In *ACM CCS 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM Press, 651–668. https://doi.org/10.1145/3319535.3354236

[16] Emma Dauterman, Henry Corrigan-Gibbs, David Mazières, Dan Boneh, and Dominic Rizzo. 2019. True2F: Backdoor-Resistant Authentication Tokens. In *2019 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 398–416. https://doi.org/10.1109/SP.2019.00048

[17] Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. 1992. Authentication and Authenticated Key Exchanges. *Designs, Codes and Cryptography* 2, 2 (June 1992), 107–125.

[18] Cynthia Dwork, Moni Naor, and Amit Sahai. 1998. Concurrent Zero-Knowledge. In *30th ACM STOC*. ACM Press, 409–418. https://doi.org/10.1145/276698.276853

[19] Florian M. Farke, Lennart Lorenz, Theodor Schnitzler, Philipp Markert, and Markus Dürmuth. 2020. "You still use the password after all" – Exploring FIDO2 Security Keys in a Small Company. In *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*. USENIX Association, 19–35. https://www.usenix.org/conference/soups2020/presentation/farke

[20] Haonan Feng, Hui Li, Xuesong Pan Pan, and Ziming Zhao. 2021. A formal analysis of the FIDO UAF protocol. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*.

[21] Nils Fleischhacker, Johannes Krupp, Giulio Malavolta, Jonas Schneider, Dominique Schröder, and Mark Simkin. 2016. Efficient Unlinkable Sanitizable Signatures from Signatures with Re-randomizable Keys. In *PKC 2016, Part I (LNCS)*, Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang (Eds.), Vol. 9614. Springer, Heidelberg, 301–330. https://doi.org/10.1007/978-3-662-49384-7_12

[22] Nick Frymann, Daniel Gardham, Franziskus Kiefer, Emil Lundberg, Mark Manulis, and Dain Nilsson. 2020. Asynchronous Remote Key Generation: An Analysis of Yubico's Proposal for W3C WebAuthn. In *ACM CCS 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM Press, 939–954. https://doi.org/10.1145/3372297.3417292

[23] Iness Ben Guirat and Harry Halpin. 2018. Formal Verification of the W3C Web Authentication Protocol. In *Proceedings of the 5th Annual Symposium and Bootcamp on Hot Topics in the Science of Security (HoTSoS '18)*. Association for Computing Machinery, New York, NY, USA, Article 6, 10 pages. https://doi.org/10.1145/3190619.3190640

[24] Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin. 2007. Stronger Security of Authenticated Key Exchange. In *ProvSec 2007 (LNCS)*, Willy Susilo, Joseph K. Liu, and Yi Mu (Eds.), Vol. 4784. Springer, Heidelberg, 1–16.

[25] Laurie Law, Alfred Menezes, Minghua Qu, Jerome A. Solinas, and Scott A. Vanstone. 2003. An Efficient Protocol for Authenticated Key Agreement. *Des. Codes Cryptogr.* 28, 2 (2003), 119–134.

[26] Sanam Ghorbani Lyastani, Michael Schilling, Michaela Neumayr, Michael Backes, and Sven Bugiel. 2020. Is FIDO2 the Kingslayer of User Authentication? A Comparative Usability Study of FIDO2 Passwordless Authentication. In *2020 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 268–285. https://doi.org/10.1109/SP40000.2020.00047

[27] Rafael Pass. 2003. On Deniability in the Common Reference String and Random Oracle Model. In *CRYPTO 2003 (LNCS)*, Dan Boneh (Ed.), Vol. 2729. Springer, Heidelberg, 316–337. https://doi.org/10.1007/978-3-540-45146-4_19

[28] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. 2001. OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption. In *ACM CCS 2001*, Michael K. Reiter and Pierangela Samarati (Eds.). ACM Press, 196–205. https://doi.org/10.1145/501983.502011

[29] Claus-Peter Schnorr. 1991. Efficient Signature Generation by Smart Cards. *Journal of Cryptology* 4, 3 (Jan. 1991), 161–174. https://doi.org/10.1007/BF00196725

[30] Isamu Teranishi, Jun Furukawa, and Kazue Sako. 2004. k-Times Anonymous Authentication (Extended Abstract). In *ASIACRYPT 2004 (LNCS)*, Pil Joong Lee (Ed.), Vol. 3329. Springer, Heidelberg, 308–322. https://doi.org/10.1007/978-3-540-30539-2_22

[31] Bitcoin Wiki. 2018. BIP32 proposal. https://en.bitcoin.it/wiki/BIP_0032. (2018).

[32] Yubico. 2020. Yubikey U2F Key Generation. https://developers.yubico.com/U2F/Protocol_details/Key_generation.html. (2020). [Online; accessed 14-January-2022].

A OMITTED PROOFS FOR KEY WRAPPING

Proof of Lemma 4.3. We show the statement by presenting a sequence of games $\mathbf{G}_i$, where for each such game $\mathbf{G}_i$ the probability that the game outputs 1 is denoted by $\mathbf{Adv}_i$.

Game $\mathbf{G}_0$: This is the real man-in-the-middle game. At the beginning of this game, every token $T \in \mathcal{T}$ is initialized with a master secret key $\mathsf{msk}_T \leftarrow \mathsf{Gen}(\mathsf{param})$. Then, the adversary gets access to oracles $\mathsf{Start}, \mathsf{Challenge}, \mathsf{Complete}$. By definition, we have

$$\mathbf{Adv}_0 = \mathbf{Adv}^{\mathcal{A}}_{\mathbf{MiM},\mathsf{kwrPA}}.$$

Game $\mathbf{G}_1$: We change the game as follows. The game is as $\mathbf{G}_0$, but it aborts if for $x \neq x'$ we have $H(x) = H(x')$. The images of H are sampled uniformly at random from $\{0,1\}^{2\lambda}$, which implies that

$$|\mathbf{Adv}_0 - \mathbf{Adv}_1| \leq \frac{Q_H^2}{2^{2\lambda}}.$$

Game $\mathbf{G}_2$: This game is as $\mathbf{G}_2$, but we introduce another abort. To this end, consider the server-side oracles Start. Recall that during the execution of such an oracle, a random $rs \xleftarrow{\$} \{0,1\}^{\geq\lambda}$ is sampled. Game $\mathbf{G}_2$ aborts if the same rs is sampled in two different invocations of the oracle Start. Clearly, we have the bound

$$|\mathbf{Adv}_1 - \mathbf{Adv}_2| \leq \frac{Q_S^2}{2^{\lambda}}.$$

Game $\mathbf{G}_3$: In this game, we will no longer generate master secret keys msk_T for each token. Recall that these keys are used in the previous games to encrypt signing keys via $\mathsf{cid} \leftarrow \mathsf{Enc}(\mathsf{msk}_T, (\mathsf{id}_S, \mathsf{sk}))$ in queries of the form $\mathsf{Challenge}(\pi_T^{i,0}, \mathsf{id}_S, \cdot)$ (i.e. in registration) and to decrypt such cid in queries of the form $\mathsf{Challenge}(\pi_T^{i,j}, \mathsf{id}_S, \mathsf{cid}, \cdot)$, $j > 0$ (i.e. in authentication). In game $\mathbf{G}_2$, we instead hold an initially empty map $L[\cdot, \cdot]$. In each registration query $\mathsf{Challenge}(\pi_T^{i,0}, \mathsf{id}_S, \cdot)$ the value cid is now sampled uniformly at random from the ciphertext space, and an entry $L[T, \mathsf{cid}] := (\mathsf{id}_S, \mathsf{sk})$ is added. In each authentication query $\mathsf{Challenge}(\pi_T^{i,j}, \mathsf{id}_S, \mathsf{cid}, \cdot)$, $j > 0$, the entry $(\mathsf{id}, \mathsf{cid}) := L[T, \mathsf{cid}]$ is retrieved from the map instead of decrypting cid, and it is used if it is defined. If it is undefined, the oracle aborts its execution.

We can bound the distinguishing advantage between $\mathbf{G}_2$ and $\mathbf{G}_3$ using $|\mathcal{T}|$ intermediate hybrids. Namely, in hybrid i, we apply the change to the first i tokens $T \in \mathcal{T}$. For each hybrid step we can give a straight-forward reduction $\mathcal{B}$ from the anonymous authentication security of ske. Thus, we have

$$|\mathbf{Adv}_1 - \mathbf{Adv}_2| \leq |\mathcal{T}| \cdot \mathbf{Adv}^{\mathcal{B}}_{\mathbf{anon\text{-}auth},\mathsf{ske}}.$$

Now, if we take a look at $\mathbf{G}_3$, we see that each challenge that the adversary gets via oracle Start and has to sign is unique. Also, signing keys sk are only needed to sign, and not in the plain anymore. Thus, similar to the proof of Lemma 4.1 we can build a reduction C from the euf-cma security of sig to bound $\mathbf{Adv}_3$. On a high level, the reduction guesses in which query of the form $\mathsf{Challenge}(\pi_T^{i,0}, \cdot)$ the adversary obtained the public key, for which it forges a signature. We have

$$\mathbf{Adv}_3 \leq Q_C \cdot \mathbf{Adv}^{C}_{\mathsf{euf\text{-}cma},\mathsf{sig}}.$$

□

Proof of Lemma 4.4. We show the claim by presenting a sequence of games $\mathbf{G}_i$. For each game $\mathbf{G}_i$, we denote the probability that it outputs 1 by pr_i.

Game $\mathbf{G}_0$: This game is the real unlinkability game. In this game, a key $\mathsf{sk}_T \leftarrow \mathsf{Gen}(\mathsf{param})$ is generated for each token $T \in \mathcal{T}$. Then, the adversary $\mathcal{A}$ gets access to oracles Start, Challenge, Complete and outputs tokens T_0, T_1 and servers S_L, S_R. Afterwards, it also gets access to oracles Left, Right, which run $\mathsf{Challenge}(\pi_{T_b}^{i_b, j_b}, \mathsf{id}_{S_L}, \cdot, \cdot)$ and $\mathsf{Challenge}(\pi_{T_{1-b}}^{i_{1-b}, j_{1-b}}, \mathsf{id}_{S_R}, \cdot, \cdot)$, respectively. By definition, we have

$$\mathsf{pr}_0 = \mathbf{Adv}^{\mathcal{A}}_{\mathbf{Unl},\mathsf{kwrPA}}.$$

Game $\mathbf{G}_1$: We change game $\mathbf{G}_0$ in the following way. At the beginning, $\mathbf{G}_1$ samples $T_0^*, T_1^* \xleftarrow{\$} \mathcal{T}$. Later, if $T_0 \neq T_0^*$ or $T_1 \neq T_1^*$, the game returns a random bit. Otherwise, it continues as $\mathbf{G}_0$ does. As $\mathcal{A}$ obtains no information about T_0^*, T_1^* until the potential abort, we have

$$\left|\mathsf{pr}_1 - \frac{1}{2}\right| = \frac{1}{|\mathcal{T}|^2} \left|\mathsf{pr}_0 - \frac{1}{2}\right|.$$

Game $\mathbf{G}_2$: We change $\mathbf{G}_1$ in the following way. Recall that in $\mathbf{G}_1$, whenever $\mathcal{A}$ starts a registration interaction with token $T_b, b \in \{0, 1\}$ via oracles Challenge or Left, Right, a ciphertext cid is created as $\mathsf{cid} := \mathsf{Enc}(\mathsf{msk}_{T_b}, (\mathsf{id}_S, \mathsf{sk}))$. Furthermore, when $\mathcal{A}$ starts an authentication interaction with token $T_b, b \in \{0, 1\}$ via oracles Challenge or Left, Right, it provides a ciphertext cid, which is then decrypted as $(\mathsf{id}, \mathsf{sk}) := \mathsf{Dec}(\mathsf{msk}, \mathsf{cid})$. If Dec returns $\perp$ or id does not match with the server identity id_S provided, the oracles abort. Otherwise, they continue their execution using secret key sk.

Now, in game $\mathbf{G}_2$, we change this encryption and decryption for the tokens T_0^*, T_1^* that we guessed in $\mathbf{G}_1$. Note that if $\mathbf{G}_1$ does not abort, we know that $(T_0^*, T_1^*) = (T_0, T_1)$ and these tokens are also used in oracles Left and Right. Concretely, the game works as follows: Initially, two empty maps $L_0[\cdot], L_1[\cdot]$ are initialized. Then, in each registration interaction with token $T_i^*, i \in \{0, 1\}$ (including the ones in oracle Left or Right) the value cid is sampled randomly from the ciphertext space. Then, an entry $L_i[\mathsf{cid}] := (\mathsf{id}_S, \mathsf{sk})$ is added. Furthermore, in each authentication interaction with token $T_i^*, i \in \{0, 1\}$ (including the ones in oracle Left or Right), where the adversary provides cid, we check if $L_i[\mathsf{cid}]$ is defined. If it is, we use it instead of decrypting cid. If it is not defined, we abort this interaction. A direct reduction $\mathcal{B}$ from the anonymous authentication security of ske shows that

$$|\mathsf{pr}_1 - \mathsf{pr}_2| \leq \mathbf{Adv}^{\mathcal{B}}_{\mathbf{anon\text{-}auth},\mathsf{ske}}.$$

We note that now, the only dependence of $\mathcal{A}$'s view on bit b is the shared use of the maps L_0, L_1. Namely, the table L_b is used by challenge oracles for token T_b and by the oracle Left.

Game $\mathbf{G}_3$: This game is as game $\mathbf{G}_2$, but we partition map L_b into two tables: The first map, L_b', is used in oracle Challenge for token T_b. The second map, L_L is used in oracle Left. The maps are used as before, and the difference is that oracle Challenge never accesses L_L and oracle Left never accesses L_b'.

We claim that the view of $\mathcal{A}$ does not change from $\mathbf{G}_2$ to $\mathbf{G}_3$. To see this, note that $\mathcal{A}$ can only observe the change, if it sends a value cid to one oracle in authentication (e.g. Challenge), which was given out by the other oracle (e.g. Left) in registration. But then, due to registration uniqueness, the oracle used for authentication would have aborted in $\mathbf{G}_2$ as well. It follows that

$$\mathsf{pr}_2 = \mathsf{pr}_3.$$

Game $\mathbf{G}_4$: This game is as $\mathbf{G}_3$, but we partition the map L_{1-b} into two tables L_{1-b}' and L_R. The change is similar as above, and the same argument shows

$$\mathsf{pr}_3 = \mathsf{pr}_4.$$

We note that in $\mathbf{G}_4$, the view of $\mathcal{A}$ is independent of bit b, which implies that $\mathsf{pr}_4 = 1/2$. □

B OMITTED PROOFS FOR BIP32

Lemma B.1. *Let* $\mathsf{sig} = (\mathsf{kg}, \mathsf{sign}, \mathsf{srerand}, \mathsf{prerand}, \mathsf{ver})$ *be a the ECDSA signature scheme and* $\tilde{H} : \{0,1\}^* \to \{0,1\}^\lambda$ *be a random oracle. For an adversary* $\mathcal{A}$*, consider the following game:*

1. *Run* $\mathcal{A}^{\tilde{H}}$ *on input* param *and obtain a string* $id \in \{0,1\}^*$ *from* $\mathcal{A}^{\tilde{H}}$.
2. *Generate fresh key pair* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{kg}(\mathsf{param})$ *and give it (including* sk*) to* $\mathcal{A}^{\tilde{H}}$.
3. *When* $\mathcal{A}^{\tilde{H}}$ *outputs* (pk', ch)*, output* 1 *if and only if*

$$\mathsf{prerand}(\mathsf{pk}', \tilde{H}(\mathsf{pk}', ch, id)) = \mathsf{pk}.$$

Then, if algorithm $\mathcal{A}$ *makes at most* $Q_{\tilde{H}}$ *queries to* $\tilde{H}$*, the probability that the game outputs* 1 *is at most* $Q_{\tilde{H}}/2^\lambda$.

Proof. Consider an adversary $\mathcal{A}$ and the game as in the statement. Let $\mathsf{sk}' \in \mathbb{Z}_p$ be such that $g^{\mathsf{sk}'} = \mathsf{pk}'$. Then, note that the winning condition is equivalent to

$$\tilde{H}(\mathsf{pk}', ch, id) = \mathsf{sk} - \mathsf{sk}'.$$

Note that pk' uniquely determines sk' and thus $\mathsf{sk} - \mathsf{sk}'$. Hence, for each random oracle query of $\mathcal{A}$, the probability that it can be used for a valid output is at most $1/2^{\lambda}$. The claim follows from a union bound. □

Proof of Lemma 6.2. We show the statement via a sequence of games. For each game $\mathbf{G}_i$, we denote the advantage of $\mathcal{A}$, i.e. probability that $\mathbf{G}_i$ outputs 1, by $\mathbf{Adv}_i$.

Game $\mathbf{G}_0$: This is the real revocation soundness game. Recall that in the beginning of the game, a key $\mathsf{msk}_T = (\mathsf{sk}_{T,0}, \mathsf{pk}_{T,0}, ch_T)$ is generated for each token. Then, the adversary gets access to oracles Start, Challenge, Complete and outputs a token T^* and a server S^*. Using these, the game executes a registration of token T^* at server S^*. Afterwards, the adversary $\mathcal{A}$ gets all information about that registration, namely $(c, M_r, \mathsf{cid}, R_r, b_r, \mathsf{cred})$. Again, $\mathcal{A}$ has access to the oracles Start, Challenge, Complete and finally outputs a revocation key rk^*. The game outputs 1 if and only if $\mathsf{CheckCred}(\mathsf{id}_{S^*}, \mathsf{cred}, \mathsf{rk}^*)$. By definition, we have

$$\mathbf{Adv}_1 = \mathbf{Adv}^{\mathcal{A}}_{\mathbf{rev\text{-}sound},\mathsf{PIANRK}}.$$

Game $\mathbf{G}_1$: This game is as $\mathbf{G}_0$, but we introduce a bad event and let the game abort if this event occurs. Consider the registration of T^* at S^*. In this registration, the algorithm rresponse is executed. We say that the bad event occurs, if the value $\hat{H}(\mathsf{pk}_{T^*,0}, ch_{T^*}, \mathsf{cid}, \mathsf{id}_{S^*})$ is already defined. Note that the value cid is sampled uniformly at random from $\{0,1\}^{\lambda}$ in rresponse. Thus, the probability that the hash value is already defined is at most $Q_{\hat{H}}/2^{\lambda}$. Therefore, we have

$$|\mathbf{Adv}_0 - \mathbf{Adv}_1| \leq \frac{Q_{\hat{H}}}{2^{\lambda}}.$$

Game $\mathbf{G}_2$: This game is as $\mathbf{G}_1$, but we change the registration of T^* at S^* again. Recall that in game $\mathbf{G}_1$, during that registration, a key pair $(\mathsf{sk}, \mathsf{pk})$ is generated using $\rho := \hat{H}(\mathsf{pk}_{T^*,0}, ch_{T^*}, \mathsf{cid}, \mathsf{id}_{S^*})$ and

$$\mathsf{sk} := \mathsf{srerand}(\mathsf{sk}_{T^*,0}, \rho),\ \mathsf{pk} := \mathsf{prerand}(\mathsf{pk}_{T^*}, \rho).$$

Especially, due to the change in $\mathbf{G}_1$, the value ρ is uniformly random at this point. In game $\mathbf{G}_2$, we sample $(\mathsf{sk}, \mathsf{pk})$ as a fresh key pair via $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{kg}(\mathsf{param})$ and program

$$\hat{H}(\mathsf{pk}_{T^*,0}, ch_{T^*}, \mathsf{cid}, \mathsf{id}_{S^*}) := \mathsf{sk} - \mathsf{sk}_{T^*,0}.$$

As the distribution of rerandomized keys is the same as fresh keys if the randomness ρ is uniform, it follows that the view of $\mathcal{A}$ does not change. Therefore, we have

$$\mathbf{Adv}_1 = \mathbf{Adv}_2.$$

Game $\mathbf{G}_3$: This game is as $\mathbf{G}_2$, but we add another bad event and make the game abort if this event occurs. Namely, we say that the bad event occurs, if $\mathcal{A}$ queries $\hat{H}(\mathsf{pk}_{T^*,0}, ch_{T^*}, \cdot, \cdot)$ at some point during the game. Clearly, all information that $\mathcal{A}$ obtains about ch_{T^*} are the random oracle hashes that it (implicitly) sees. Thus, by a union bound over $\mathcal{A}$'s random oracle queries we get

$$|\mathbf{Adv}_2 - \mathbf{Adv}_3| \leq \frac{Q_{\hat{H}}}{2^{\lambda}}.$$

Game $\mathbf{G}_4$: This game is as $\mathbf{G}_3$, but we add another bad event and make the game abort if this event occurs. We say that the bad event occurs if $\mathcal{A}$'s final output is $\mathsf{rk}^* = (\mathsf{pk}_{T^*,0}, ch_{T^*})$. As all information that $\mathcal{A}$ obtains about ch_{T^*} are the random oracle hashes that it sees, the probability of this bad event is at most $1/2^{\lambda}$. Thus,

$$|\mathbf{Adv}_3 - \mathbf{Adv}_4| \leq \frac{1}{2^{\lambda}}.$$

Finally, we can bound the advantage $\mathbf{Adv}_4$ of $\mathcal{A}$ in game $\mathbf{G}_4$ using a reduction from the game in Lemma B.1. The reduction gets as input parameters param and gets oracle access to an oracle $\tilde{H}$. It sets up the game for $\mathcal{A}$ as in $\mathbf{G}_4$, while simulating oracle $\hat{H}$ by forwarding queries to $\tilde{H}$. Then, once the adversary outputs T^*, S^*, the reduction runs $c \leftarrow \mathsf{rchallenge}(\mathsf{id}_{S^*}), M_r \leftarrow \mathsf{rcommand}(\mathsf{id}_{S^*}, c)$ as in $\mathbf{G}_4$. Then, it samples $\mathsf{cid} \xleftarrow{\$} \{0,1\}^{\lambda}$ and outputs $id := (\mathsf{cid}, \mathsf{id}_{S^*})$ to its game. It obtains $(\mathsf{sk}, \mathsf{pk})$ from its game and programs

$$\hat{H}(\mathsf{pk}_{T^*,0}, ch_{T^*}, \mathsf{cid}, \mathsf{id}_{S^*}) := \mathsf{sk} - \mathsf{sk}_{T^*,0}$$

as in $\mathbf{G}_3$. Later, when $\mathcal{A}$ outputs $\mathsf{rk}^* = (\mathsf{pk}', ch)$, it outputs (pk', ch) to its own game.

It is easy to see that the reduction perfectly simulates game $\mathbf{G}_4$ for $\mathcal{A}$. Moreover, if the bad event defined in $\mathbf{G}_4$ does not occur, the random oracles $\tilde{H}$ and $\hat{H}$ coincide on (pk', ch, id), which is why the reduction wins its game. Thus, Lemma B.1 implies that $\mathbf{Adv}_4 \leq Q_{\hat{H}}/2^{\lambda}$ and the statement follows. □

Proof of Lemma 6.3. We note that the proof is very similar to the proof of Lemma 4.2, and we use some parts literally.

We show the statement via a sequence of games. For each game $\mathbf{G}_i$, we denote the probability that $\mathbf{G}_i$ outputs 1 by pr_i.

Game $\mathbf{G}_0$: This game is the real unlinkability game. At the beginning of the game, a master secret key $\mathsf{msk}_T = (\mathsf{sk}_{T,0}, \mathsf{pk}_{T,0}, ch_T)$ is generated for each token $T \in \mathcal{T}$. The adversary $\mathcal{A}$ gets access to oracles Start, Challenge, Complete. Then, it outputs two tokens T_0, T_1 and servers S_L, S_R. Afterwards, it also gets access to oracles Left, Right, which internally run $\mathsf{Challenge}(\pi_{T_b}^{i_b,j_b}, \mathsf{id}_{S_L}, \cdot, \cdot)$ and $\mathsf{Challenge}(\pi_{T_{1-b}}^{i_{1-b},j_{1-b}}, \mathsf{id}_{S_R}, \cdot, \cdot)$, respectively. Additionally, the adversary gets access to random oracle $\hat{H}$. By definition, we have

$$\mathsf{pr}_0 = \mathbf{Adv}^{\mathcal{A}}_{\mathbf{Unl},\mathsf{bip32PA}}.$$

Game $\mathbf{G}_1$: This game is as $\mathbf{G}_0$, but we introduce a bad event and let the game abort if this bad event occurs. The bad event occurs, if the adversary ever queries $\hat{H}(\mathsf{pk}_{T,0}, ch_T, \cdot, \cdot)$ for any token $T \in \mathcal{T}$. Note that $\mathcal{A}$ obtains no information about ch_T throughout the game (except via hash values). Thus, we can use a union bound over the $Q_{\hat{H}}$ random oracle queries and the $|\mathcal{T}|$ tokens to obtain

$$\left|\mathsf{pr}_0 - \mathsf{pr}_1\right| \leq \frac{Q_{\hat{H}} \cdot |\mathcal{T}|}{2^{\lambda}}.$$

Game $\mathbf{G}_2$: This game is as $\mathbf{G}_1$, but we introduce another bad event and let the game abort if this bad event occurs. The bad event occurs, if there are tokens $T \neq T'$ such that $ch_T = ch_{T'}$. Clearly, we have

$$\left|\mathsf{pr}_1 - \mathsf{pr}_2\right| \leq \frac{|\mathcal{T}|^2}{2^{\lambda}}.$$

Game $\mathbf{G}_3$: This game is as $\mathbf{G}_2$, but we change the way oracle Left works. Recall that in input cid, M, oracle $\mathsf{Left}(\mathsf{cid}, M)$ in game

$\mathbf{G}_2$ samples a key pair $(\mathsf{sk}, \mathsf{pk})$ as follows: It derives randomness $\rho := \hat{H}(\mathsf{pk}_{T_b,0}, ch_{T_b}, \mathsf{cid}, \mathsf{id}_{S_L})$ and sets

$$\mathsf{sk} := \mathsf{srerand}(\mathsf{sk}_{T_b,0}, \rho),\ \mathsf{pk} := \mathsf{prerand}(\mathsf{pk}_{T_b,0}, \rho).$$

Now, assuming the game $\mathbf{G}_3$ does not abort, we know that value ρ is uniformly distributed for $\mathcal{A}$ here. Especially, due to registration uniqueness and non-colliding ch's, the oracle Challenge never queries $\hat{H}(\mathsf{pk}_{T_b,0}, ch_{T_b}, \mathsf{cid}, \mathsf{id}_{S_L})$. As the distributions of rerandomized key pairs is the same as the one of fresh key pairs, we can change sk, pk to a fresh key pair. To be precise, game $\mathbf{G}_3$ keeps a map $K_L[\cdot]$, and in each call $\mathsf{Left}(\mathsf{cid}, M)$ it first checks if $K_L[\mathsf{cid}]$ is already defined. If it is defined, it is used as a key pair. Otherwise, a fresh key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{kg}(\mathsf{param})$ is sampled and stored as $K_L[\mathsf{cid}] := (\mathsf{sk}, \mathsf{pk})$. The rest of the oracle stays the same, using that particular key pair. The above argument shows that if the game does not abort, the view of $\mathcal{A}$ does not change. Thus we have

$$\mathsf{pr}_2 = \mathsf{pr}_3.$$

Game $\mathbf{G}_4$: This game is as $\mathbf{G}_3$, but we change how oracle Right behaves. Concretely, we apply a similar change using fresh key pairs and a map K_R as we did for oracle Left in game $\mathbf{G}_3$. With the same arguments, it follows that

$$\mathsf{pr}_3 = \mathsf{pr}_4.$$

Finally, note that oracles Left and Right in game $\mathbf{G}_4$ and thus $\mathcal{A}$'s view are independent of the bit b. Thus, we have $\mathsf{pr}_4 = 1/2$ and the claim follows. □

Token(msk)		**Client**(id_S)		**Server**(id_S)
$\underline{(\mathrm{cid}, R_r) \leftarrow \mathrm{rresponse}(\mathrm{msk}, \mathrm{id}_S, M_r):}$		$\underline{M_r \leftarrow \mathrm{rcommand}(\mathrm{id}_S, c):}$		$\underline{c \leftarrow \mathrm{rchallenge}(\mathrm{id}_S):}$
$h := M_r, \mathrm{cid} \overset{\$}{\leftarrow} \{0,1\}^\lambda$		$(\mathrm{id}, rs) := c$		$rs \overset{\$}{\leftarrow} \{0,1\}^{\geq\lambda}$
$\mathrm{sk} := KDF(\mathrm{msk}, \mathrm{cid}, \mathrm{id}_S), \mathrm{pk} := \mathrm{toPK}(\mathrm{sk})$ / kdfPA	$\xleftarrow{\mathrm{id}_S, M_r}$	**if** $\mathrm{id} \neq \mathrm{id}_S$: **abort**	$\xleftarrow{c}$	$c := (\mathrm{id}_S, rs)$
$(\mathrm{sk}, \mathrm{pk}) \leftarrow \mathrm{kg}(\mathrm{param})$ / kwrPA		$M_r := H(rs)$		
$\mathrm{cid} := \mathrm{Enc}(\mathrm{msk}, (\mathrm{id}_S, \mathrm{sk}))$ / kwrPA				
$(\mathrm{sk}_0, \mathrm{pk}_0, ch) := \mathrm{msk}$ / bip32PA				$\underline{(b, \mathrm{cred}) \leftarrow \mathrm{rcheck}(\mathrm{cid}, R_r)}$
$\mathrm{sk} := \mathrm{srerand}(\mathrm{sk}_0, \hat{H}(\mathrm{pk}_0, ch, \mathrm{cid}, \mathrm{id}_S))$ / bip32PA				$m := (H(\mathrm{id}_S), \mathrm{cid}, \mathrm{pk}, H(rs))$
$\mathrm{pk} := \mathrm{prerand}(\mathrm{pk}_0, \hat{H}(\mathrm{pk}_0, ch, \mathrm{cid}, \mathrm{id}_S))$ / bip32PA				$b := \mathrm{ver}(\mathrm{pk}, \sigma, m)$
$\sigma \leftarrow \mathrm{sign}(\mathrm{sk}, (H(\mathrm{id}_S), \mathrm{cid}, \mathrm{pk}, h))$	$\xrightarrow{\mathrm{cid}, R_r}$		$\xrightarrow{\mathrm{cid}, R_r}$	**if** $b = 0$: $\mathrm{cred} := \perp$
$R_r := (\mathrm{pk}, \sigma)$				$\mathrm{rcs}[\mathrm{cid}] := \mathrm{pk}, \ \mathrm{cred} := (\mathrm{cid}, \mathrm{pk}))$

Figure 2: The WebAuthn registration protocol. The highlighted statements are only executed in the variation that is given in the respective comment. Functions V_t, V_s (cf. Definition 3.4) are given as $(H(\mathrm{id}_S), H(rs), \mathrm{cid})$.

Token(msk)		**Client**(id_S)		**Server**(id_S, cid)
$\underline{R_a \leftarrow \mathrm{aresponse}(\mathrm{msk}, \mathrm{id}_S, \mathrm{cid}, M_a):}$		$\underline{M_a \leftarrow \mathrm{acommand}(\mathrm{id}_S, c):}$		$\underline{c \leftarrow \mathrm{achallenge}(\mathrm{id}_S):}$
$\mathrm{sk} := KDF(\mathrm{msk}, \mathrm{cid}, \mathrm{id}_S)$ / kdfPA	$\xleftarrow{\mathrm{id}_S, \mathrm{cid}, M_a}$	$(\mathrm{id}, rs) := c$	$\xleftarrow{\mathrm{cid}, c}$	$rs \overset{\$}{\leftarrow} \{0,1\}^{\geq\lambda}$
$(\mathrm{id}, \mathrm{sk}) := \mathrm{Dec}(\mathrm{msk}, \mathrm{cid})$ / kwrPA		**if** $\mathrm{id} \neq \mathrm{id}_S$: **abort**		$c := (\mathrm{id}_S, rs)$
if $\mathrm{id} \neq \mathrm{id}_S$: **abort** / kwrPA		$M_a := H(rs)$		
$(\mathrm{sk}_0, \mathrm{pk}_0, ch) := \mathrm{msk}$ / bip32PA				$\underline{b \leftarrow \mathrm{acheck}(\mathrm{cid}, R_a)}$
$no \overset{\$}{\leftarrow} \{0,1\}^\lambda, \ M_a := (M_a, no)$ / bip32PA				$\mathrm{pk} \leftarrow \mathrm{rcs}[\mathrm{cid}]$
$\mathrm{sk} := \mathrm{srerand}(\mathrm{sk}_0, \hat{H}(\mathrm{pk}_0, ch, \mathrm{cid}, \mathrm{id}_S))$ / bip32PA				$m := (H(\mathrm{id}_S), H(rs)),$
$\sigma \leftarrow \mathrm{sign}(\mathrm{sk}, (H(\mathrm{id}_S), M_a))$	$\xrightarrow{R_a}$		$\xrightarrow{R_a}$	$m := (H(\mathrm{id}_S), H(rs), no)$ / bip32PA
$R_a := \sigma, R_a := (\sigma, no)$ / bip32PA				$b := \mathrm{ver}(\mathrm{pk}, \sigma, m)$

Figure 3: The WebAuthn authentication protocol. The highlighted statements are only executed in the variation that is given in the respective comment. Functions V_t, V_s (cf. Definition 3.4) are given as $(H(\mathrm{id}_S), H(rs), \mathrm{cid})$.