Token meets Wallet: Formalizing Privacy and Revocation for FIDO2*

Lucjan Hanzlik[1], Julian Loss[1], and Benedikt Wagner[1,2]

[1]CISPA Helmholtz Center for Information Security
[2]Saarland University

Abstract

The FIDO2 standard is a 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) It only covers tokens that store *all secret keys locally*. In contrast, due to limited memory, most existing FIDO2 tokens either derive all secret keys from a common seed or store keys on the server (the latter approach is also known as *key wrapping*).

In this paper, we revisit the security of the WebAuthn component of 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) We provide the *first formal definition of privacy for the WebAuthn component of FIDO2*. We then prove the privacy of this component in common FIDO2 token implementations if the underlying building blocks are chosen appropriately. (3) We address the unsolved problem of *global key revocation* in FIDO2. To this end, we introduce and analyze a simple revocation procedure that builds on the popular BIP32 standard used in cryptocurrency wallets and can efficiently be implemented with existing FIDO2 servers.

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 to use a device (*something she has*) that is registered to the user's account. Universal Second Factor (U2F) (or CTAP1, as it is currently named) is a protocol to achieve two-factor authentication using a designated device that we refer to as a *token*. The token runs a simple piece of code and interacts with the user's interface, e.g., a web browser. We refer to this interface as the *client* or *agent*. The agent acts as a proxy device during the authentication process between a token and a server that we call the *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 specification [fid19]. Together with the complement W3C web authentication [web20] (WebAuthn) they form the state-of-the-art for online authentication using security tokens.

In recent work, Barbosa et al. [BBCW21] gave the first formal model for token-based authentication and provided a security proof for the FIDO2 standard. While their model provides an important starting point for further exploration of FIDO2's security properties, it does not accurately model several key aspects of FIDO2 as

*This is the full version of a paper with the same title at IEEE S&P 2023.

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. Before presenting our contributions in more detail, we first explain several disparities between the model of Barbosa et al. and how FIDO2 is commonly used in practice.

Attestation. Barbosa et al. consider a model where the token and the relying party are initially provided a pair of attestation secret key and attestation public key by the manufacturer. Thereby, servers can verify that they interact with tokens from trusted manufacturers. 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 [BCZ22]. Also, naive attestation violates privacy [GH18]. Therefore, to reason about the security and privacy of FIDO2 in its most widely used form, we consider a model *without attestation*.

Privacy. There are two ways to implement FIDO2 tokens. The first is to store all secret keys of the token locally, i.e. on the authentication token. In practice however, keys are almost exclusively stored *externally* on the server [KHWK22, Yub20]. External key storage can be implemented securely using so called *key wrapping*, where an encryption of the key is stored with the server. Another approach is to use a *key derivation function* to derive keys on the fly. Throughout, we refer to both key wrapping and key derivation function approaches as *external key storage*.

Since tokens are limited with regards to storage space, externally stored keys provide a distinct storage advantage over locally stored keys. On the downside, externally stored keys may give the server additional information that it may use to link separate sessions of the token.

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. Intuitively, in the random oracle model, the above scheme is as secure as Enc. However, ciphertexts produced by the same token (i.e. using the same key k) can trivially be linked using the second component.

The above discussion shows that the use of externally stored 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 locally stored keys, their model does not provide any treatment of privacy. Thus, proving the privacy properties claimed by FIDO2 remains an open problem.

1.1 Our Contribution

In this work, we present a new security model for the WebAuthn component of FIDO2. We analyze existing FIDO2 implementations in this model. In a second step, we identify a lack of practical key revocation in these implementations that is compatible with our privacy goals. We now explain our contributions in more detail.

New Security Models for Privacy and Impersonation. Our first contribution is to augment the model of Barbosa et al. with a suitable notion of privacy, and analyzing existing token implementations using key wrapping and key derivation functions in this augmented model. Concretely, we define a natural security game with three different winning conditions, leading to *strong, medium and weak unlinkability*. Intuitively, the winning condition for strong unlinkability only disallows a generic linking attack that no token implementation can mitigate [KHWK22]. For medium and weak unlinkability, we slightly weaken the restrictions on this attack. The results of our analysis in these models are given in the top part of Table 1. Notably, while key wrapping provides strong unlinkability if we rely on *anonymous* authenticated encryption, our analysis shows that implementations using the key derivation function approach provide only medium unlinkability. In this sense, our privacy notions show exactly which linking attacks are possible for these implementation, and reveal a gap between the key wrapping and key derivation approaches for externally storing keys. On top of these new privacy definitions, we also define an appropriate notion of security against impersonation in the context of externally stored keys and show that existing implementations satisfy it.

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. [LSN+20] shows that one

of the top concerns of users is that an adversary can gain access to their account in case their FIDO2 token gets lost or is stolen. For all current token implementations, a user has to revoke all keys separately by logging in at each server, which is highly impracticable. To address this revocation problem in a usable way, we desire a procedure that allows to simultaneously revoke *all keys* associated with an token without communicating with each server individually. This includes also any future keys that the (compromised) token may attempt to register with the relying party. Ideally, revocation is done by globally publishing some short revocation information, which is then noticed by all servers. The revocation information can be stored externally (e.g. on a piece of paper) by the user in a secure location. We refer to such a procedure as *global revocation*. In terms of security, we desire three properties: First, as long as the revocation information is not published, all credentials should be unlinkable to guarantee privacy. Second, we demand security against impersonation even after publishing the revocation information. This is of great importance, as otherwise an attacker could actively look for such published revocation information and impersonate a user before a server that periodically checks for revocations may notice. Finally, it should be difficult for a malicious party to launch a denial of service (DoS) attack in which it revokes keys on behalf of an honest user. In Section 6 we formalize these security properties of global revocation.

As we observe, however, achieving these properties for global revocation is a challenging endeavour. To get an intuition why, consider a naive approach where every credential of a user is prefixed with some common tag t. To revoke, one can simply publish t. This approach clearly links all of the prefixed credentials and hence violates any reasonable notion of privacy. Additionally, a DoS attack as mentioned above is possible, because t is not kept secret. This shows the technical challenge introduced by global revocation: On the one hand, keys can not be generated independently, as the published revocation information has to identify all keys. On the other hand, generating keys using some correlated randomness may violate the users privacy and security.

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, ECDSA 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 $\mathsf{sk}_{\mathsf{id}} := \mathsf{msk} + w, \mathsf{pk}_{\mathsf{id}} := \mathsf{mpk} \cdot g^w$. We observe that this procedure provides a simple means of global revocation. To revoke all public keys associated with a pair $\mathsf{msk}, \mathsf{mpk}$, one can simply publish mpk and chain. Now, each server can find deterministically recompute the public keys from mpk and chain, and revoke them. After introducing this new token implementation based on BIP32, we show that it satisfies the security and privacy properties that we defined. We highlight that 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 for key-prefixed ECDSA given that no message is ever signed twice [DEF+21]. As we want our new token implementation to be compatible with existing FIDO2 server implementations, we can not rely on key-prefixing as it is used by Das et al. [DEF+21], and have to find a way to allow signing the same message multiple times. This makes the analysis challenging. For further details, we refer the reader to Section 7. Finally, we give experimental results that showcase the efficiency of our protocol during authentication and revocation.

1.2 Related Work

The first formal model for the FIDO2 protocol was proposed by Barbosa et al. [BBCW21]. The authors introduced the notion of passwordless authentication that models the WebAuthn protocol in the scenario when the token uses keys stored locally. They also present 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 access control notion and proposes an alternative protocol based on password-authenticated key agreement (PAKE).

Bindel, Cremers, and Zhao [BCZ22] (to appear at S&P 2023) recently extended the Barbosa et al. model to support the latest specification, introducing CTAP in version 2.1. Their model uses explicit user verification

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

Scheme	Unlinkability	Authentication	Revocation
kdfPA	Medium	Secure	Local
kwrPA	Strong	Secure	Local
bip32PA	Weak	Secure	Global/Local

Table 1: Overview of our results. We give the first analysis of existing schemes kdfPA, kwrPA, and propose and analyze scheme bip32PA. In local revocation, the user has to revoke all keys separately, while global revocation means that the user can publish one string to revoke all keys simultaneously.

assurance and guarantees stronger token binding properties. Relevant to our contribution is that they consider the case where attestation of the token is not used (attestation mode `None`) and argue that this is the most widely used mode. Unfortunately, their model, similar to [BBCW21], only works if the token permanently stores all keys. They also do not consider any privacy definition that would model the desired property of cross relying party unlinkability of the token-generated credentials.

Guirat et al. [GH18] 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. The security and privacy issues faced by developers during the implementation of FIDO and WebAuthn components were discussed by Alam et al. [AKB19]. Kepkowski et al. [KHWK22] report that existing relying parties are implemented based on existing open-source solutions and, by default, support only externally stored keys. The authors also show a timing attack allowing a malicious relying party to link users across different services, proving that a formal definition and analysis of privacy is needed. Another line of related work provides alternative protocols and usability studies. Chakraborty et al. [CB19] used a TPM implementation (simTPM) based on a sim card as a secure and convenient FIDO2 token. A usability and acceptability study of FIDO2 was done by Lyastani et al. [LSN+20]. The authors report that one of the main concerns of users is that an adversary can access their account if they lose their token or it gets stolen. An efficient and usable revocation mechanism would solve some of those concerns and increase the acceptance rate among users.

Revocation in anonymous authentication and privacy-preserving signatures is not a new problem. Camenisch and Lysyanskaya [CL02] showed how cryptographic accumulators can be used to revoke users in an anonymous setting,. The system's authority provides users membership certificates, allowing them to prove zero-knowledge membership against a publicly available accumulator. Boneh and Shacham [BS04] used a different approach in the context of group signatures, the so-called verifier-local revocation. The verifier is the only party involved in the revocation process. Camenisch et al. [CDH16] discussed revocation techniques for pseudonymous systems. Their solution allows checking the revocation status in logarithmic time in the number of revoked users, independently of all valid users in the system. A simple alternative approach to VLR is blocklisting in pseudonymous systems. During the interaction, users generate a domain/application-specific unlinkable pseudonym that the verifier compares with elements on the blocklist leading to the check taking logarithmic time in the number of revoked users.

Dauterman et al. [DCM+19] discussed the problem of backdooring FIDO tokens. They introduced the notion of verifiable identity families (VIF) and used it to generate FIDO identities. The key feature is that the tokens can prove an identity (i.e. public key) to be correct (given a relying party) and pseudorandom to any party holding a VIF master public key. It allows the user to verify that the token deterministically creates identities and a manufacturer did not implement any backdoor in the key generation algorithm. Their solution does not support revocation without per-server interaction since identities are not publicly linkable. To link identities, the token must create proof of correct VIF evaluation.

Frymann et al. [FGK+20] discussed the token backup procedure proposed by Yubico, a recovery solution in case of token loss/theft. The solution allows a primary token to generate an identity that the backup device will use with a specific relying party. Thus, instead of registering both devices, a user can register once using the primary device and use the backup token if the former is lost. The problem of a backup device is orthogonal to revocation, i.e. in case the primary device gets stolen/lost, it still has to be revoked to protect the user's account.

From a cryptographic point of view, FIDO2 is a simple authentication scheme, a primitive which has been extensively studied. One of the first and well-known protocols is the Schnorr identification scheme [Sch91], which provides impersonation resistance under the discrete logarithm assumption. Authenticated key exchange (AKE) protocols provide means to agree on a secret key between the interacting parties and simultaneously authenticate both parties [DvW92, LMQ+03, LLM07]. A different approach to authentication is the folklore challenge-response protocol based on signature schemes, on which FIDO2 is based. This technique provides a framework for creating authentication protocols. Instantiated with group signatures [Cv91, BMW03] the protocol provides a way for group members to authenticate to a server anonymously. A direct solution introduced by Teranishi, Furukawa, and Sako is anonymous authentication [TFS04]. The extended access control protocol (EAC) [DF11, BF17] is an alternative approach to authentication using a hardware device for machine-readable travel documents (e.g., e-Passports).

2 Notation and Preliminaries

We denote by $z \leftarrow \mathcal{A}(x)$ the execution of algorithm $\mathcal{A}$ on input x and with output z. If we want to make the random coins ρ of algorithm $\mathcal{A}$ explicit, we write $z := \mathcal{A}(x;\rho)$ instead. 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 $\{1,\ldots,n\}$. By λ, we denote the security parameter. Throughout the paper, we assume public parameters par are given implicitly to all algorithms. We will use standard notions of digital signatures and one of the rerandomizable signature schemes discussed in [FKM+16, DFL19]. To formally model the key wrapping technique, we will use a definition of symmetric key encryption that is both authenticated and anonymous [RS06]. We present their syntax and security definitions in more detail in Appendix A.

3 WebAuthn and its Implementations

In this section, we give an overview of the WebAuthn protocol contained in the FIDO2 standard, and introduce the syntax we need for our formal analysis. FIDO2 can be used as means of passwordless authentication or as a second-factor for the standard login-via-password scenario. The WebAuthn protocol template consists of a registration and an authentication process executed between a token and a server. In both registration and authentication, the protocol consists of two messages sent between the server and the token, relayed through a client which is implemented e.g. by the user's browser. The first message is sent from the server to the token (via the client) and contains a challenge rs. This challenge is then signed by the token, and the resulting signature σ is sent back to the server. In the registration protocol, this second message also contains a public key pk, and a so called *credential identifier* cid. The server stores this key and additional information in a *credential* cred. In subsequent authentication interactions, the server first finds cid (e.g. by looking up the username), and then includes it in the first message of the above flow.

Different token implementations may generate key material in a different way. To accommodate for memory-constraints, the most commonly used implementations do not store the key material locally, but instead use the value cid to securely outsource it to the server. In this paper we will focus on the following two variants:

- *Key Derivation Function.* The signing key is generated in a pseudorandom way using a master secret key, the server's identifier, and the randomly chosen credential identifier. 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 the credential identifier. This method is e.g. used by Yubico in their implementation of FIDO2 tokens [Yub20].

We will now describe the above two schemes in more detail. An overview (in our syntax) can be found in Figure 1. We refer to the two variants as kdfPA and kwrPA, respectively. WebAuthn makes use of a hash function $H : \{0,1\}^* \to \{0,1\}^{2\lambda}$, modeled as a random oracle.

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

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 with length function $\nu : \mathbb{N} \to \mathbb{N}$. Thus, we set $\mathsf{msk} \leftarrow \mathsf{SKE.Gen}(\mathsf{par})$.

Registration. We assume that every server S holds a registration context rcs_S, which stores users credentials. During registration, the server S generates a random nonce $rs \xleftarrow{\$} \{0,1\}^\lambda$ and sends the challenge $c = (\mathsf{id}_S, rs)$ to the client, where id_S is a server identifier. The client verifies that id_S is correct and sends $M_r = H(rs)$ and id_S to the token. Here, the token implementations kwrPA and kdfPA differ slightly. In kdfPA, the token first chooses a random identifier $\mathsf{cid} \xleftarrow{\$} \{0,1\}^\lambda$. It then uses a pseudorandom key derivation function PRF to generate a secret key $\mathsf{sk} := \mathsf{PRF}(\mathsf{msk}, (\mathsf{cid}, \mathsf{id}_S))$. We assume that PRF outputs values in the secret key space of the signature scheme. The token computes the corresponding public key $\mathsf{pk} := \mathsf{ToPK}(\mathsf{sk})$. In kwrPA the key pair is computed using the signature scheme key generation algorithm $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(\mathsf{par})$. The secret key sk and id_S is then encrypted by the token as $\mathsf{cid} := \mathsf{Enc}(\mathsf{msk}, (\mathsf{id}_S, \mathsf{sk}))$. We assume that pairs $(\mathsf{sk}, \mathsf{id}_S)$ of signing secret keys and server identifiers all have the same length $l_0 = l_0(\lambda) \in \mathbb{N}$, which is achieved using appropriate padding. We set $\nu^* := \nu(l_0)$. For both schemes the secret key sk is used to create the signature $\sigma \leftarrow \mathsf{Sig}(\mathsf{sk}, m)$, where $m := (H(\mathsf{id}_S), \mathsf{cid}, \mathsf{pk}, M_r)$. Finally, the token sends a response message $R_r = (\mathsf{pk}, \sigma)$ together with cid to the client, which forwards it to the server. The server verifies the token's response as follows. It aborts and stores no credential in case $\mathsf{Ver}(\mathsf{pk}, \sigma, m) = 0$, where $m := (H(\mathsf{id}_S), \mathsf{cid}, \mathsf{pk}, M_r)$. Otherwise, it accepts. This means that it adds pk to its registration context, i.e. $\mathsf{rcs}_S[\mathsf{cid}] := \mathsf{pk}$.

Authentication. For authentication, we assume that the server knows the credential identifier cid of interest. Finding cid depends on the concrete use case of WebAuthn, e.g. using FIDO2 as a second factor, or in a standalone fashion. For example, a user could enter its username, and the server holds a mapping from usernames to credential identifiers. The authentication protocol begins with the server generating a random nonce $rs \xleftarrow{\$} \{0,1\}^\lambda$ and sending the challenge $c = (\mathsf{id}_S, rs)$ and cid to the client. The client verifies that id_S is correct and sends the message $M_a := H(rs)$, the server identifier id_S and identifier cid to the token. Now, in the first step the token recreates the signing key sk that it created during registration. For kdfPA this means that the token runs $\mathsf{sk} := \mathsf{PRF}(\mathsf{msk}, (\mathsf{cid}, \mathsf{id}_S))$. 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 $\mathsf{id} \neq \mathsf{id}_S$. For both schemes the secret key sk is used to create the signature $\sigma \leftarrow \mathsf{Sig}(\mathsf{sk}, m)$, where $m := (H(\mathsf{id}_S), M_a)$. Finally, the token sends $R_a = \sigma$ to the client, which forwards it to the server. The server verifies the token's response as follows. First, the server uses the registration context rcs to get the token's public key $\mathsf{pk} \leftarrow \mathsf{rcs}_S[\mathsf{cid}]$. Then, it sets $m := (H(\mathsf{id}_S), H(rs))$ and accepts if $\mathsf{Ver}(\mathsf{pk}, \sigma, m) = 1$.

Formal Syntax. In our model, we consider parties $\mathcal{P} = \mathcal{T} \cup \mathcal{S}$, partitioned into the set of tokens $\mathcal{T}$ and the set of servers $\mathcal{S}$. Each server $S \in \mathcal{S}$ keeps as a registration context rcs_S as internal state, which is an initially empty key-value table. When new tokens register, an entry to rcs_S is added. Each token $T \in \mathcal{T}$ keeps a fixed state that is initialized once with a key msk_T and does not change. We do not model clients explicitly. Instead, we allow the adversary to do the computation that the clients do. As in [BBCW21], we assume that each server has a unique identifier id_S. In reality, id_S corresponds to a URL, which justifies this assumption. In all experiments that we define, we assume that these identifiers are given to all parties. We do not explicitly model the initial contact of user and server, as this may differ depending on the use case. That means that we assume that the server already knows the users account and therefore its cid.

Definition 1 (Passwordless Authentication)**.** *A* passwordless authentication scheme *(PlA) is a tuple* $\mathsf{PLA} = (\mathsf{Gen}, \mathsf{Reg}, \mathsf{Auth})$ *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 algorithms* $(\mathsf{rchall}, \mathsf{rcomm}, \mathsf{rresp}, \mathsf{rcheck})$.
  - *The randomized* registration challenge generation algorithm rchall *takes as input a server identity* id_S *and outputs a challenge value* c *and a state* st.

- *The deterministic* registration command creation algorithm rcomm *takes as input a server identity* id_S *and a challenge value* c *and outputs a message* M_r.
- *The randomized* registration response algorithm rresp *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 state* st*, 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 *is given as a tuple of algorithms* $(\mathsf{achall}, \mathsf{acomm}, \mathsf{aresp}, \mathsf{acheck})$.
  - *The randomized* authentication challenge generation algorithm achall *takes as input a server identity* id_S *and outputs a challenge value* c *and a state* st.
  - *The deterministic* authentication command creation algorithm acomm *takes as input a server identity* id_S *and a challenge value* c *and outputs a message* M_a.
  - *The randomized* authentication response algorithm aresp *takes as input a master secret key* msk*, a server identity* id_S*, a credential identifier* cid*, and a message* M_a *and outputs a response* R_a.
  - *The deterministic* authentication check algorithm acheck *takes as input a state* st*, a registration context* rcs*, a credential identifier* cid *and a response* R_a *and outputs a bit* $b \in \{0,1\}$

Algorithms $\mathsf{rchall}, \mathsf{rcheck}, \mathsf{achall}, \mathsf{acheck}$ *are executed by servers,* $\mathsf{rcomm}, \mathsf{acomm}$ *are executed by clients, and* $\mathsf{rresp}, \mathsf{aresp}$ *are executed by tokens.*

Definition 2 (Completeness of PIA)**.** *We say that a PIA* $\mathsf{PLA} = (\mathsf{Gen}, \mathsf{Reg}, \mathsf{Auth})$ *with* $\mathsf{Reg} = (\mathsf{rchall}, \mathsf{rcomm}, \mathsf{rresp}, \mathsf{rcheck})$ *and* $\mathsf{Auth} = (\mathsf{achall}, \mathsf{acomm}, \mathsf{aresp}, \mathsf{acheck})$ *is complete, if for all* $\mathsf{msk} \in \mathsf{Gen}(\mathsf{par})$*, parties* T *and* S*, sets* R_{init}, R_{betw} *of tuples* $(\mathsf{cid}, \mathsf{cred})$*, the probability that the following experiment outputs* 0 *is* 0*:*

1. *Let* rcs *be a key-value table. For each* $(\mathsf{cid}, \mathsf{cred}) \in R_{init}$*, set* $\mathsf{rcs}[\mathsf{cid}] := \mathsf{cred}$.
2. *Run the registration protocol* Reg *of* T *at* S*, as follows:*

$$(c, \mathsf{st}) \leftarrow \mathsf{rchall}(\mathsf{id}_S), \quad M_r \leftarrow \mathsf{rcomm}(\mathsf{id}_S, c),$$
$$(\mathsf{cid}, R_r) \leftarrow \mathsf{rresp}(\mathsf{msk}, \mathsf{id}_S, M_r), \quad (b_r, \mathsf{cred}) \leftarrow \mathsf{rcheck}(\mathsf{st}, \mathsf{cid}, R_r).$$

   If $b_r = 0$*, output* 0*. Otherwise, set* $\mathsf{rcs}[\mathsf{cid}] := \mathsf{cred}$.
3. *For each* $(\mathsf{cid}, \mathsf{cred}) \in R_{betw}$*, set* $\mathsf{rcs}[\mathsf{cid}] := \mathsf{cred}$.
4. *Run the authentication protocol* Auth *of* T *at* S*, which is as follows:*

$$(c, \mathsf{st}) \leftarrow \mathsf{achall}(\mathsf{id}_S), \quad M_a \leftarrow \mathsf{acomm}(\mathsf{id}_S, c),$$
$$R_a \leftarrow \mathsf{aresp}(\mathsf{msk}, \mathsf{id}_S, \mathsf{cid}, M_a), \quad b_a \leftarrow \mathsf{acheck}(\mathsf{st}, \mathsf{rcs}, \mathsf{cid}, R_a).$$

5. *Return* b_a.

We assume that cid *derived in Step 2 is not in the list* R_{betw}.

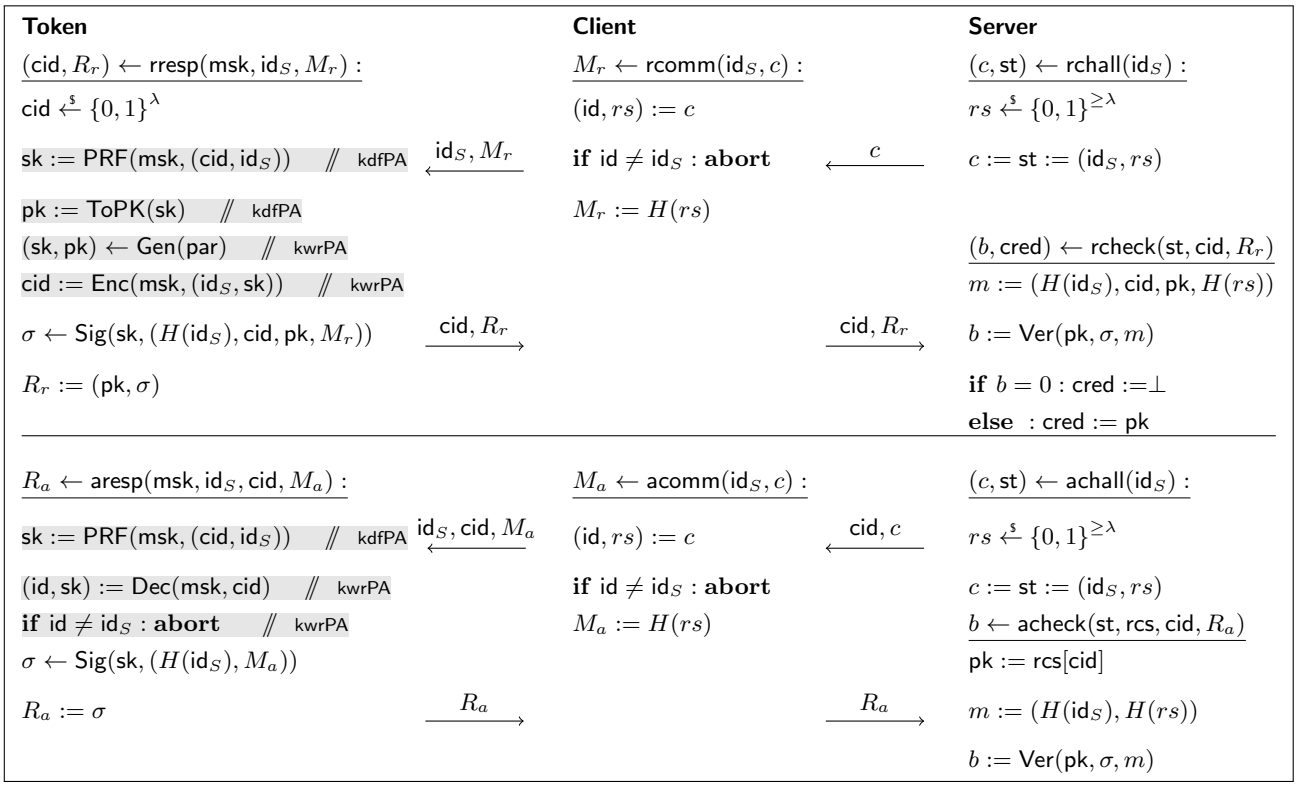

Figure 1: The WebAuthn registration (top) and authentication protocol (bottom) protocol for the variations kdfPA and kwrPA. The highlighted statements are only executed in the variation that is given in the respective comment. Functions V_t, V_s (cf. Definition 4) are given as $(H(\mathsf{id}_S), H(rs), \mathsf{cid})$.

4 Modeling Security and Privacy

In this section, we formally define security and privacy models for WebAuthn. While privacy is not considered in [BBCW21], our security model is closely related to their model. Before presenting our models, 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, and will be used in all of our security definitions. For server S, rcs_S is the registration context, i.e the set of credentials it stores, st_S is used to model the state transferred between algorithms rchall, achall and rcheck, acheck, respectively, and the map C_S is used for bookkeeping to bind registration interactions to authentication interactions.

Definition 3 (Server and Token Oracles)**.** *Let $\mathcal{A}$ be an algorithm and* $\mathsf{PLA} = (\mathsf{Gen}, \mathsf{Reg}, \mathsf{Auth})$ *be a PIA. We associate each party $P \in \mathcal{T} \cup \mathcal{S}$ with a set of handles $\pi_P^{i,j}$ which model two types of instances corresponding to registration and authentication. Each party is represented by a number of these instances. Concretely, we refer to $\pi_P^{i,j}$ for $j = 0$ as the ith registration instance of party P and for $j \geq 1$ as the jth authentication instance of P corresponding to the ith registration.*

We assume that for each token $T \in \mathcal{T}$, a secret key $\mathsf{msk}_T \leftarrow \mathsf{Gen}(\mathsf{par})$ *is given, and that for each server $S \in \mathcal{S}$, key-value tables* rcs_S, C_S, *and* st_S *are given. Per default, these are empty. Then, adversary $\mathcal{A}$ has access to oracles* Start, Challenge, Complete *as follows.*

- $\mathsf{Start}(\pi_S^{i,j})$*: This executes* $(c, \mathsf{st}) \leftarrow \mathsf{rchall}(\mathsf{id}_S)$ *in case* $j = 0$ *or* $(c, \mathsf{st}) \leftarrow \mathsf{achall}(\mathsf{id}_S)$ *in case* $j > 0$*. The oracle sets* $\mathsf{st}_S[i, j] := \mathsf{st}$*, and returns c to $\mathcal{A}$.*
- $\mathsf{Challenge}(\pi_T^{i,j}, \mathsf{id}_S, \mathsf{cid}, M)$*: This runs algorithm* $(\mathsf{cid}, R_r) \leftarrow \mathsf{rresp}(\mathsf{msk}_T, \mathsf{id}_S, M)$ *if* $j = 0$ *or* $R_a \leftarrow$

aresp($\mathsf{msk}_T, \mathsf{id}_S, \mathsf{cid}, M$) *if* $j > 0$. *The result (*(cid, R_r) *or* R_a*) is returned to* $\mathcal{A}$. *Note that the input* cid *is ignored for* $j = 0$.

- Complete($\pi_S^{i,j}, \mathsf{cid}, R$)*: This aborts if* Start($\pi_S^{i,j}$) *has not been queried before. If* $j = 0$*, it runs* $(b, \mathsf{cred}) \leftarrow$ rcheck($\mathsf{st}_S[i,j], \mathsf{cid}, R$)*, sets* $C_S[i] := \mathsf{cid}$*, and* $\mathsf{rcs}_S[\mathsf{cid}] := \mathsf{cred}$. *If* $j > 0$*, it aborts if* $\mathsf{cid} \neq C_S[i]$. *Otherwise, it runs* $b \leftarrow$ acheck($\mathsf{st}_S[i,j], \mathsf{rcs}_S, \mathsf{cid}, R$). *In both cases,* b *is returned to* $\mathcal{A}$.

We assume that for each $(i, j, T, S) \in \mathbb{N} \times \mathbb{N} \times \mathcal{T} \times \mathcal{S}$*, the oracles* Start($\pi_S^{i,j}$), Challenge($\pi_T^{i,j}, \cdot, \cdot, \cdot$)*, and* Complete($\pi_S^{i,j}, \cdot, \cdot$) *are executed only once.*

The index i in token handles $\pi_T^{i,j}$ may seem artificial at first, as tokens are stateless. However, this index will simplify the following definition significantly. Namely, we follow the work of Bellare et al. [BPR00] to define partnering of handles, which will be used in the winning condition of security experiments. Two handles $\pi_S^{i,j}$ and $\pi_T^{i',j'}$ are partnered if they share the same session identifier. One should think of partnered handles as if the adversary just forwarded messages between the respective oracles. Formally, the notion of session identifiers must be defined by the protocol. A natural choice is that the session identifiers correspond to the view of a party. Note that the choice of session identifiers influences the notion of security that is achieved.

Definition 4 (Session Identifiers and Partnering)**.** *Let* PLA *be a PIA and consider the oracles from Definition 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* Challenge($\pi_T^{i,j}, \cdot, \cdot, \cdot$)*, 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* Start($\pi_S^{i,j}$), Complete($\pi_S^{i,j}, \cdot, \cdot$)*, and outputs a bitstring* $V_s(tr_S^{i,j})$*. We assume that these functions are specified by* PLA.

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

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

4.1 Impersonation Security

We now define what it means for a passwordless authentication protocol to be secure against impersonation. Informally, we say that if this property holds then the token that registered must be used to authenticate against a server and a single interaction cannot be used to authenticate multiple times. In the security game, the adversary can interact with tokens and servers in an arbitrary way, using the oracles from Definition 3.

Definition 5 (Impersonation Security)**.** *For PIA* PLA $=$ (Gen, Reg, Auth) *and an adversary* $\mathcal{A}$*, we define the experiment* $\mathbf{Imp}_{\mathsf{PLA}}^{\mathcal{A}}$ *as follows.*

- **Setup.** *For each token* $T \in \mathcal{T}$*, a key is generated by running* $\mathsf{msk}_T \leftarrow$ Gen(par).
- **Online Phase.** *The adversary is allowed to interact with the oracles* Start, Challenge, Complete *as in Definition 3.*
- **Output Phase.** *Finally,* $\mathcal{A}$ *terminates, and the experiment outputs* 1 *if and only if there exists a server handle* $\pi_S^{i,j}$ *for* $j > 0$ *such that the following conditions hold:*
  1. $\pi_S^{i,0}$ *is partnered with a token handle* $\pi_T^{k,0}$.
  2. $\pi_S^{i,j}$ *accepted, i.e. in call* Complete($\pi_S^{i,j}, \mathsf{cid}, R$)*, algorithm* acheck($\mathsf{st}_S[i,j], \mathsf{rcs}_S, \mathsf{cid}, R$) *returned* 1.
  3. $\pi_S^{i,j}$ *is not partnered with any token handle* $\pi_T^{i',j'}$*, or it is partnered with a token handle, which is partnered with a different server handle* $\pi_{S'}^{i'',j''}$.

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

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

Let us explain the three winning conditions that are defined. By the first condition, we know that the token T registered at server S. By the second and the third condition we know that the adversary was able to authenticate at server S for that particular registration. We highlight that this models a "trust-on-first-use" concept. Namely, servers consider the user communicating during registration as the honest user. In addition to the stateless nature of tokens, this is another difference to the model by Barbosa et al. [BBCW21], who use attestation keys as their trust anchor.

4.2 Unlinkability

We define unlinkability for passwordless authentication. Informally, we want that interactions of the same token are unlinkable, and different registrations of the same token can not be linked. 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.

The Unlinkability Experiment. To model the above goal in a security experiment, we let the adversary $\mathcal{A}$ interact with all oracles in Definition 3 to get a global view of the system. Then, $\mathcal{A}$ targets two tokens T_0, T_1, and gets two additional oracles Left, Right, running T_b and T_{1-b} internally, for a random bit b. The goal of $\mathcal{A}$ is to determine which token is used in which oracle. That means that if $\mathcal{A}$ can link interactions with the same token, it can for example link the interaction with oracle Left to the interaction with token oracle Challenge for token T_b. This would allow him to win the game. Also, note that unlinkability can only make sense if there are at least two tokens used in the system, which is the reason why $\mathcal{A}$ has to output two tokens.

To prevent trivial linking, we have to introduce two winning conditions. The first one is a consequence of our definition of oracles and ensures that $\mathcal{A}$ can not violate the assumption in Definition 3. For the second condition, namely credential separation, we define three variants, leading to strong, medium, and weak unlinkability guarantees. Due to the stateless nature of tokens, the behavior of a token is determined by cid and id_S. This leads to the following trivial attack: Adversary $\mathcal{A}$ gets a cid from oracle Left during registration, and submits it to oracle Challenge for token T_0 and server identity id_{S_L}. It compares the resulting view (e.g. returned public keys) with the view that it gets from submitting cid to oracle Left in authentication. If the views are consistent, T_0 is used in Left. Our strongest notion of credential separation, leading to *strong unlinkability*, rules out exactly this behavior of $\mathcal{A}$.

In practice, servers can follow this strategy to determine if any credential cid in its database belongs to a user that aims to log in. The server just sends cid instead of the credential that is actually stored for this user and checks if the authentication goes through. It is worth noting that such an attack can be launched almost unnoticed by the user. In recent work, Kepkowski et al. [KHWK22] showed a timing attack against unlinkability where they exploit a small time difference on some hardware tokens when handling the cid. Their paper's key idea is to use the WebAuthn specification to amplify the attack. To support multiple tokens, the relying party sends a list of supported cid's, which are locally queried to the token by the browser to find the correct one, i.e. the one for which the token responds with a valid signature. Those queries do not require interaction and are oblivious to the user.

Going back to our example, the malicious relying party can add cid to the list containing the correct credential for the user account. If the response is a valid signature for cid, the relying party knows that cid corresponds to this account. If this distinguishing fails, the user is oblivious to the attack.

Weaker Notions. Not every implementation used in practice is strongly unlinkable. As we still aim for an analysis and a precise statement about which attacks allow to link, we define weaker variations of the above notion. To define *medium unlinkability*, we make the credential separation condition more restrictive. Namely, we also rule out that $\mathcal{A}$ queries the same cid at oracle Challenge (for the target $T_0, T_1, \mathsf{id}_{S_L}, \mathsf{id}_{S_R}$) and Left, Right, *even if the*

cid *is made up by $\mathcal{A}$ and not returned in a registration of these oracles*. Looking ahead, this subtle difference will be required to prove unlinkability in case cid is not authenticated. For *weak unlinkability*, we additionally rule out the case that oracle Challenge (for the target $T_0, T_1, \mathsf{id}_{S_L}, \mathsf{id}_{S_R}$) and Left, Right *return the same* cid *during registration*. If cid has high enough entropy this implies medium unlinkability. However, there are schemes only satisfying weak unlinkability.

Definition 6 (Unlinkability)**.** *For a PIA* PLA $=$ (Gen, Reg, Auth) *and an adversary $\mathcal{A}$, we define experiments $\mathbf{wUnl}, \mathbf{mUnl}, \mathbf{sUnl}$ as follows.*

- **Setup.** *For each token $T \in \mathcal{T}$, a key is generated by running* $\mathsf{msk}_T \leftarrow \mathsf{Gen}(\mathsf{par})$.
- **Phase 1.** *The adversary is allowed to interact with oracles* Start, Challenge, Complete *(see Definition 3).*
- **Phase 2.** *The adversary outputs two (not necessarily distinct) token identifiers T_0, T_1, and two (not necessarily distinct) server identifiers $S_L, S_R \in \mathcal{S}$. Let i_0 and i_1 be the smallest identifiers for which the token handles $\pi_{T_0}^{i_0,0}$ and $\pi_{T_1}^{i_1,0}$ were not queried to the* Challenge *oracle in Phase 1. The experiment chooses a bit b uniformly at random. It sets $j_0 := 0, j_1 := 0$ and initializes two oracles* Left, Right *as follows:*
  - $\mathsf{Left}(\mathsf{cid}, M)$*: Return* $\mathsf{Challenge}(\pi_{T_b}^{i_b,j_b}, \mathsf{id}_{S_L}, \mathsf{cid}, M)$ *and set $j_b = j_b + 1$.*
  - $\mathsf{Right}(\mathsf{cid}, M)$*: Return* $\mathsf{Challenge}(\pi_{T_{1-b}}^{i_{1-b},j_{1-b}}, \mathsf{id}_{S_R}, \mathsf{cid}, M)$ *and set $j_{1-b} = j_{1-b} + 1$.*
- **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 $\hat{b}$. Consider the following lists of* cid*'s:*
  - $\mathcal{L}_{ch}^r$ *contains all* cid*'s returned by queries that are not issued via* Left, Right *and are of the form* $\mathsf{Challenge}(\pi_T^{i,0}, \mathsf{id}_S, \cdot, \cdot)$ *for any $i, T \in \{T_0, T_1\}$ and $S \in \{S_L, S_R\}$.*
  - $\mathcal{L}_{ch}^a$ *contains all* cid*'s that are part of the input of queries that are not issued via* Left, Right *and are of the form* $\mathsf{Challenge}(\pi_T^{i,j}, \mathsf{id}_S, \cdot, \cdot)$ *for any $j > 0$,$i, T \in \{T_0, T_1\}$ and $S \in \{S_L, S_R\}$.*
  - $\mathcal{L}_{lr}^r$ *contains all* cid*'s returned by queries to* Left *or* Right *when $j_b = 0$ or $j_{1-b} = 0$, respectively.*
  - $\mathcal{L}_{lr}^a$ *contains all* cid*'s that are part of the queries to* Left *or* Right *when $j_b > 0$ or $j_{1-b} > 0$, respectively.*

  The experiment outputs a random bit z if one of the following two conditions is violated:
  - *(instance freshness) the adversary never made a query to oracle* Challenge *using handles $\pi_{T_0}^{i_0,k_0}$ and $\pi_{T_1}^{i_1,k_1}$ for any k_0, k_1.*
  - *(credential separation) The following set is empty:*

$$\begin{aligned} \mathbf{sUnl}: &\qquad S_{\mathbf{sUnl}} := (\mathcal{L}_{ch}^r \cap \mathcal{L}_{lr}^a) \cup (\mathcal{L}_{lr}^r \cap \mathcal{L}_{ch}^a)\,, \\ \mathbf{mUnl}: &\qquad S_{\mathbf{mUnl}} := S_{\mathbf{sUnl}} \cup (\mathcal{L}_{ch}^a \cup \mathcal{L}_{lr}^a), \\ \mathbf{wUnl}: &\qquad S_{\mathbf{wUnl}} := S_{\mathbf{mUnl}} \cup (\mathcal{L}_{ch}^r \cup \mathcal{L}_{lr}^r). \end{aligned}$$

  The experiment returns 1 if and only if bit $\hat{b}$ is equal to bit b.

For $x \in \{\mathbf{w}, \mathbf{m}, \mathbf{s}\}$, we define the advantage of $\mathcal{A}$ in winning the experiment as:

$$\mathbf{Adv}_{x\mathbf{Unl},\mathsf{PLA}}^{\mathcal{A}} := \left| \Pr[x\mathbf{Unl}_{\mathsf{PLA}}^{\mathcal{A}} = 1] - \frac{1}{2} \right|.$$

The credential separation condition models the attacks a server can do when the same token is used twice at the same server. Even for schemes with strong unlinkability, there is a trivial active attack, as sketched above. For schemes with weak unlinkability, but no medium unlinkability, there is a passive attack that only involves the two registrations.

5 Analysis of Existing Implementations

In this section, we analyze the existing token implementations kwrPA and kdfPA of WebAuthn. Recall that these schemes are formally given in Figure 1. We analyze both impersonation security and unlinkability.

Analysis of Impersonation Security. As the first part of our analysis, we show that both implementations kwrPA and kdfPA satisfy impersonation security. We obtain the following statements.

Lemma 1. *Let $\mathcal{A}$ be an adversary in the impersonation game of* kdfPA. *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 exists algorithms $\mathcal{B}, \mathcal{B}'$ with the same running time as $\mathcal{A}$ such that*

$$\mathbf{Adv}^{\mathcal{A}}_{\mathbf{Imp},\mathsf{kdfPA}} \leq \frac{Q_S^2 + Q_C^2}{2^\lambda} + \frac{Q_H^2}{2^{2\lambda}} + |\mathcal{T}| \cdot \mathbf{Adv}^{\mathcal{B}'}_{\mathbf{prf},\mathsf{PRF}} + Q_C \cdot \mathbf{Adv}^{\mathcal{B}}_{\mathbf{euf\text{-}cma},\mathsf{SIG}}$$

Lemma 2. *Let $\mathcal{A}$ be an adversary in the impersonation game of* kwrPA. *Assume that $\mathcal{A}$ makes at most Q_H queries to random oracle H, and at most Q_S, Q_C queries to oracles* Start, Challenge, *respectively. Then there exist algorithms $\mathcal{B}$ and $\mathcal{C}$ with the same running time as $\mathcal{A}$ such that*

$$\mathbf{Adv}^{\mathcal{A}}_{\mathbf{Imp},\mathsf{kwrPA}} \leq \frac{Q_S^2}{2^\lambda} + \frac{Q_H^2}{2^{2\lambda}} + |\mathcal{T}| \cdot \mathbf{Adv}^{\mathcal{B}}_{\mathbf{anon\text{-}auth},\mathsf{SKE}} + Q_C \cdot \mathbf{Adv}^{\mathcal{C}}_{\mathbf{euf\text{-}cma},\mathsf{SIG}}.$$

We give a proof intuition here, and postpone the formal proofs to Appendices B.1 and B.2.

Proof Intuition. Our goal is to give a reduction from the $\mathbf{euf\text{-}cma}$ security of SIG. To do that, recall that the adversary wins the impersonation game, if a token first registers a public key pk at a server, and then the adversary authenticates, i.e. it forges a valid signature for $m = (H(\mathsf{id}_S), H(rs))$ with respect to pk, without using the token. We call this interaction the forged authentication.

Our strategy is to embed the public key that we get from the $\mathbf{euf\text{-}cma}$ game into one of the registration interactions between oracle Challenge and the adversary. Secret keys are not only used to sign, and we need to deal with that. Namely, in variant kdfPA, we first need to apply pseudorandomness of PRF to make secret keys independent and random. The variant kwrPA additionally outputs a ciphertext cid encrypting the secret key. To eliminate this dependency on the secret key, we perform a hybrid step to switch all cid's output by oracle Challenge to random, while internally storing a mapping from cid to the secret key to ensure consistency.

Now we can give a reduction that first guesses the registration interaction in which it embeds the given public key. To simulate the signatures for the embedded public key, the reduction can use the signing oracle provided by the $\mathbf{euf\text{-}cma}$ game. Finally, it can use the signature that the adversary sends in the forged authentication as a forgery for the $\mathbf{euf\text{-}cma}$ game. To see that the forgery is fresh, we rule out collisions for random oracle H and the challenges rs that servers send. □

Analysis of Unlinkability. Next, we give our results in terms of unlinkability. Notably, we obtain strong unlinkability for kwrPA, while kdfPA only satisfies medium unlinkability.

Lemma 3. *Let $\mathcal{A}$ be an adversary in the medium unlinkability game of* kdfPA. *Assume that $\mathcal{A}$ makes at most Q_C queries to oracle* Challenge. *Then, there is an algorithm $\mathcal{B}$, which has the same running time as $\mathcal{A}$, such that*

$$\mathbf{Adv}^{\mathcal{A}}_{\mathbf{mUnl},\mathsf{kdfPA}} \leq \frac{2Q_C}{2^\lambda} + |\mathcal{T}| \cdot \mathbf{Adv}^{\mathcal{B}}_{\mathbf{prf},\mathsf{PRF}}.$$

Lemma 4. *Let $\mathcal{A}$ be an adversary in the strong unlinkability game of* kwrPA*. Then there exists an algorithm $\mathcal{B}$, which has the same running time as $\mathcal{A}$, such that*

$$\mathbf{Adv}^{\mathcal{A}}_{\mathbf{sUnl},\mathsf{kwrPA}} \leq |\mathcal{T}|^2 \cdot \left(\mathbf{Adv}^{\mathcal{B}}_{\mathbf{anon\text{-}auth},\mathsf{SKE}} + \frac{2Q_C}{2^{\nu^*}} \right).$$

Again, we give a proof intuition here, and postpone formal proofs to Appendices B.1 and B.2.

Proof Intuition. To show unlinkability, our goal is to remove all information about tokens T_b, T_{1-b} from the oracles Left and Right. For the variant kdfPA and medium unlinkability, this can be done roughly as follows. We first use the entropy of msk_{T_b} to argue that the adversary can only access the prefixed random oracle $G := K(\mathsf{msk}_{T_b}, \cdot)$ via oracles Challenge and Left. Then, we rule out that two registrations sample the same cid. This essentially tells us that we can assume weak credential separation instead of medium credential separation. Using weak credential separation, we see that oracles Challenge and Left access oracle G on distinct points, which allows us to split the oracle into two oracles. Using a similar argument for T_b and Right, we can conclude.

For kwrPA, our strategy is different. Namely, we first guess the target tokens and then apply the security of the encryption scheme, switching all cid's for these tokens to random. As in the proof of impersonation security, we hold a mapping to keep the keys that we use consistent. Now, for different cid's the tokens use independent keys. The only way to link between oracles Challenge and Left, Right, is to reuse values cid. This is forbidden by the strong credential separation condition. In contrast to the medium credential separation condition, the adversary is allowed to submit a fresh cid that is not output in an registration to Challenge and Left, Right. However, as the encryption scheme is authenticated, this will lead to aborting tokens. □

To see why kdfPA does not achieve strong unlinkability in general, consider an adversary that implements the following strategy. It first chooses two target tokens T_0, T_1 and server $S_L = S_R$ passes them to the experiment. Then, it runs honest registrations with oracle Left, oracle Right, and the oracle Challenge for token T_0. It then samples some random cid, and submits it during authentication interactions with these three oracles, using the same challenge M. All oracles will return valid signatures. Assuming that these are deterministically computed or reveal the public key, the adversary can simply compare the public keys to link. This attack would not be possible if cid's were authenticated, e.g. using a MAC. In this case, we are confident that one could prove medium unlinkability.

6 Defining 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 PLA. 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, and should be stored externally, e.g. on a piece of paper. Recall from Definition 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 $\mathsf{cred} = \mathsf{rcs}[\mathsf{cid}]$ in its registration context. If this algorithm accepts then the credential is considered revoked.

Definition 7 (Global Key Revocation)**.** *A PIA* $\mathsf{PLA} = (\mathsf{Gen}, \mathsf{Reg}, \mathsf{Auth})$ *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.
- 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 $\mathsf{msk} \in \mathsf{Gen}(\mathsf{par})$*, parties* T *and* S*, sets* R_{init}, R_{betw} *of tuples* $(\mathsf{cid}, \mathsf{cred})$ *the following experiment outputs* 1 *with probability* 1*:*

1. *Run steps (1)-(3) from the experiment in Definition 2.*
2. *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. In the security experiment we define, the adversary gets arbitrary access to token and server oracles. Then, it has to choose two partnered oracles corresponding to a registration. It gets the corresponding credential and has to output a revocation key to revoke it. Intuitively, this means that it tries to revoke an honest registration, with which it can arbitrarily interfere.

Definition 8 (Revocation Soundness)**.** *Let* $\mathsf{PLA} = (\mathsf{Gen}, \mathsf{Reg}, \mathsf{Auth})$ *be a PIA. Consider an algorithm* $\mathcal{A}$ *and the following experiment* $\mathbf{rev\text{-}sound}^{\mathcal{A}}_{\mathsf{PLA}}$*:*

- **Setup.** *For each token* $T \in \mathcal{T}$*, a key is generated by running* $\mathsf{msk}_T \leftarrow \mathsf{Gen}(\mathsf{par})$.
- **Online Phase.** $\mathcal{A}$ *gets access to oracles* Start, Challenge, Complete *as in Definition 3.*
- **Output Phase.** $\mathcal{A}$ *outputs tuples* (T^*, i_{T^*}) *and* (S^*, i_{S^*})*. If the oracle* $\mathsf{Complete}(\pi^{i_{S^*},0}_{S^*}, \cdot, \cdot)$ *has never been queried, the experiment returns* 0*. Otherwise, the experiment returns* cred^* *to* $\mathcal{A}$*, where* cred^* *is the credential that oracle* $\mathsf{Complete}(\pi^{i_{S^*},0}_{S^*}, \cdot, \cdot)$ *added to* rcs_{S^*}*. Then,* $\mathcal{A}$ *outputs* rk^**. The experiment outputs* 1 *if and only if* $\pi^{i_{T^*},0}_{T^*}$ *and* $\pi^{i_{S^*},0}_{S^*}$ *are partnered and* $\mathsf{CheckCred}(\mathsf{id}_{S^*}, \mathsf{cred}^*, \mathsf{rk}^*) = 1$.

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

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

Even for globally revoked keys impersonation should be impossible, in case a server does not update its state in time. Therefore, we extend the notion of impersonation security.

Definition 9 (Impersonation Security - GR)**.** *Let* $\mathsf{PLA} = (\mathsf{Gen}, \mathsf{Reg}, \mathsf{Auth})$ *be a PIA. We consider the experiment given in Definition 5 with the following modification, and call the resulting experiment* $\mathbf{Imp\text{-}GR}^{\mathcal{A}}_{\mathsf{PLA}}$*: 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*

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

For unlinkability, it is clear that all keys of a token can be linked once the revocation key is published. Thus, in our modified unlinkability experiment, the adversary gets all revocation keys except the ones for the challenge tokens. As our scheme only satisfies the weak version of unlinkability with global revocation, we only define this. The medium and strong version can be defined analogously.

Definition 10 (Unlinkability - GR)**.** *Let* $\mathsf{PLA} = (\mathsf{Gen}, \mathsf{Reg}, \mathsf{Auth})$ *be a PIA. We consider the experiment* $\mathbf{wUnl}$ *given in Definition 6 with the following modification, and call the resulting experiment* $\mathbf{wUnl\text{-}GR}^{\mathcal{A}}_{\mathsf{PLA}}$*: 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, when* $\mathcal{A}$ *outputs* T_0, T_1 *and* S_L, S_R *in Phase 2, the experiment gives* $\{\mathsf{rk}_T\}_{T \in \mathcal{T} \setminus \{T_0, T_1\}}$ *to* $\mathcal{A}$*. The rest of the experiment is as in* $\mathbf{wUnl}$*. We define the advantage of an algorithm* $\mathcal{A}$ *as*

$$\mathbf{Adv}^{\mathcal{A}}_{\mathbf{wUnl\text{-}GR},\mathsf{PLA}} := \left| \Pr[\mathbf{wUnl\text{-}GR}^{\mathcal{A}}_{\mathsf{PLA}} = 1] - \frac{1}{2} \right|.$$

7 BIP32 Passwordless Authentication

In this section, we show how to instantiate FIDO2 using ECDSA with the BIP32 key derivation [Wik18]. The resulting scheme, denoted by bip32PA supports global key revocation.

7.1 Scheme Description

We give a description of our scheme bip32PA. Then, we also explain how it can support global revocation.

ECDSA Signatures. We recall the ECDSA signature scheme and its key-prefixed variant. The system parameters par of the scheme contain a group $\mathbb{G}$ of prime order p with generator $g \in \mathbb{G}$. Let $H' : \{0,1\}^* \to \mathbb{Z}_p$ be a random oracle. We denote the scheme by $\mathsf{SIG}_{H'} = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{SRerand}, \mathsf{PRerand}, \mathsf{Ver})$ and its key-prefixed variant as $\mathsf{SIG}^{kp}_{H'}$. We first describe key generation and randomization. Key generation and rerandomization are as follows:

- $\mathsf{Gen}(\mathsf{par})$: Sample $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_p$, $\mathsf{pk} := g^{\mathsf{sk}}$. Return $(\mathsf{sk}, \mathsf{pk})$.
- $\mathsf{SRerand}(\mathsf{sk}, \rho \in \mathbb{Z}_p)$: Return $\mathsf{sk}' := \mathsf{sk} + \rho$.
- $\mathsf{PRerand}(\mathsf{pk}, \rho \in \mathbb{Z}_p)$: Return $\mathsf{pk}' := \mathsf{pk} \cdot g^{\rho}$.

Next, let us explain signing and verification. Signing and verifcation makes use of algorithms $\mathsf{Sig}^{int}, \mathsf{Ver}^{int}$ that are used internally. For the purpose of this work, we can treat these as a black box. For details, see [DEF+21]. Using these algorithms, signing (i.e. algorithm $\mathsf{Sig}(\mathsf{sk}, m)$) is as follows:

1. Set $pm := m$ ((pk, m) for the key-prefixed variant).
2. Compute $z := H'(pm)$.
3. Return $\sigma \leftarrow \mathsf{Sig}^{int}(\mathsf{sk}, z)$.

Verification computes z in the same way and runs $\mathsf{Ver}^{int}(\mathsf{pk}, \sigma, z)$. Looking ahead, the fact that pm is hashed will allow the reductions in our proof to map key-prefixed messages to messages prefixed with a server identifier.

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

Registration and Authentication. Registration and authentication follow the WebAuthn specification. Thus, they are very similar to the protocols described in Section 5. Formally, we present protocols Reg and Auth in Figure 2. 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 defines $\mathsf{cid} := \hat{H}(seed, \mathsf{id}_S)$ using a random oracle $\hat{H} : \{0,1\}^* \to \{0,1\}^{\lambda}$. Then, it derives a randomness $\rho := \hat{H}(\mathsf{pk}_0, ch, \mathsf{id}_S)$. 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 stores $\mathsf{cid}, \mathsf{pk}$ during registration and the key sk is used to sign challenges.

Let us give a brief explanation of the design choices made here. Implementing the signing process in a provably secure way that is compatible with existing server implementations is non-trivial. This is because provable security with respect to randomized keys for ECDSA is only known for the key-prefixed version and for one signature per message [DEF+21]. To support multiple signatures per message, one idea is to let the token choose a random nonce and append it to messages. As we do not want to change the protocol on the server side, we can not rely on such random nonces. Instead, we derandomize the signing process by deriving the random coins used for signing a message m as $\hat{H}(seed, m)$. It remains to avoid key-prefixing, as this would also require changing verification on the server side. Here, we makes use of the fact that with high probability, the mapping from server identities to public keys is injective. Therefore, prefixing with server identities (which is already done in WebAuthn) is as good as prefixing with public keys. To make this idea work, we need to ensure that for each server, there is a fixed

$(\mathsf{cid}, R_r) \leftarrow \mathsf{rresp}(\mathsf{msk} = (\mathsf{sk}_0, \mathsf{pk}_0, ch, seed), \mathsf{id}_S, M_r)$:	$R_a \leftarrow \mathsf{aresp}(\mathsf{msk} = (\mathsf{sk}_0, \mathsf{pk}_0, ch, seed), \mathsf{id}_S, \mathsf{cid}, M_a)$:
$\mathsf{cid} := \hat{H}(seed, \mathsf{id}_S)$	**if** $\mathsf{cid} \neq \hat{H}(seed, \mathsf{id}_S)$: **abort**
$\mathsf{sk} := \mathsf{SRerand}(\mathsf{sk}_0, \hat{H}(\mathsf{pk}_0, ch, \mathsf{id}_S))$	$\mathsf{sk} := \mathsf{SRerand}(\mathsf{sk}_0, \hat{H}(\mathsf{pk}_0, ch, \mathsf{id}_S))$
$\mathsf{pk} := \mathsf{PRerand}(\mathsf{pk}_0, \hat{H}(\mathsf{pk}_0, ch, \mathsf{id}_S))$	$m := (H(\mathsf{id}_S), M_a),\ coins := \hat{H}(seed, m)$
$m := (H(\mathsf{id}_S), \mathsf{cid}, \mathsf{pk}, M_r),\ coins := \hat{H}(seed, m)$	$\sigma := \mathsf{Sig}(\mathsf{sk}, m; coins),\ R_a := \sigma$
$\sigma := \mathsf{Sig}(\mathsf{sk}, m; coins),\ R_r := (\mathsf{pk}, \sigma)$	

Figure 2: The WebAuthn registration (left) and authentication protocol (right) for our new variation bip32PA. Functions V_t, V_s, and algorithms rcomm, acomm, rchall, achall, rcheck, acheck are given as in Figure 1.

public key, which explains the definition of cid. As cid is deterministically derived from id_S, each registration on the same server will be associated with the same public key. This is also the reason why the scheme only achieves weak unlinkability.

Global Revocation. The advantage of the BIP32 key derivation compared to existing schemes is global key revocation, as defined in the previous section. We present algorithms Revoke and CheckCred for our scheme bip32PA. Algorithm Revoke(msk) is given as follows:

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

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

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

7.2 Revocation Soundness and Unlinkability

Next, we show revocation soundness and unlinkability of bip32PA. We postpone the formal proofs to Appendix C, and give proof intuitions here.

Revocation Soundness. We show revocation soundness. Recall that this means that only the owner of a token can revoke its keys. Our result is summarized in the following statement.

Lemma 5. *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 $\hat{H}$ and at most Q_C queries to oracle* Challenge*. Then we have*

$$\mathbf{Adv}^{\mathcal{A}}_{\mathbf{rev\text{-}sound},\mathsf{bip32PA}} \leq \frac{Q_{\hat{H}} \cdot |\mathcal{T}| + Q_{\hat{H}}^2 + 1 + Q_C \cdot Q_{\hat{H}}}{2^{\lambda}}.$$

Proof Intuition. In the experiment, the adversary first instructs a token T^* with key $\mathsf{msk}_T = (\mathsf{sk}_{T^*,0}, \mathsf{pk}_{T^*,0}, ch_{T^*}, seed_{T^*})$ and a server S^* to interact in a registration. Then, it gets the resulting credential $\mathsf{cred}^* = \mathsf{pk}$, and has to output a revocation key $\mathsf{rk}^* = (\mathsf{pk}_0^*, ch^*)$. The adversary breaks revocation soundness, if $\mathsf{CheckCred}(\mathsf{id}_{S^*}, \mathsf{cred}^*, \mathsf{rk}^*) = 1$, i.e. if $\mathsf{pk} = \mathsf{PRerand}(\mathsf{pk}_0^*, \hat{H}(\mathsf{pk}_0^*, ch^*, \mathsf{id}_{S^*}))$. To show that this is infeasible, we first argue using the entropy of ch_{T^*} that the adversary will not output $\mathsf{rk}^* = (\mathsf{pk}_{T^*,0}, ch_{T^*})$. Then, we guess the registration interaction of interest and embed an independent public key pk into this registration. We can do this by programming random oracle $\hat{H}$ accordingly. With this, winning the game reduces to solving the following isolated problem: Given a key pair $(\mathsf{sk}, \mathsf{pk})$ and access to a random oracle $\tilde{H}$, find pk' and ch such that $\mathsf{pk} = \mathsf{PRerand}(\mathsf{pk}', \tilde{H}(\mathsf{pk}', ch))$. This problem is statistically hard to solve, as we show in Appendix C, Lemma 10. □

Unlinkability. We show unlinkability in presence of global revocation. We obtain the following result.

Lemma 6. *Let $\mathcal{A}$ be an adversary in the weak unlinkability with global revocation game of* bip32PA*. Assume that $\mathcal{A}$ makes at most $Q_{\hat{H}}, Q_H$ queries to random oracles $\hat{H}, H$, respectively. Then we have*

$$\mathbf{Adv}^{\mathcal{A}}_{\mathbf{wUnl\text{-}GR},\mathsf{bip32PA}} \leq \frac{4 \cdot Q_{\hat{H}} + 2 \cdot |\mathcal{T}|^2}{2^{\lambda}} + \frac{Q_H^2}{2^{2\lambda}}.$$

Proof Intuition. The proof is similar to the proof of variant kdfPA. Namely, we first argue that the adversary can access the random oracles $G_1 := \hat{H}(\mathsf{pk}_{T_j,0}, ch_{T_j}, \cdot, \cdot)$ or $G_2 := \hat{H}(seed_{T_j}, \cdot)$ for the challenge tokens $j \in \{0,1\}$ only indirectly via the oracles Left, Right and Challenge. Then, we claim that the oracles Left and Challenge access these prefixed random oracles on distinct inputs. As we have perfect rerandomization of keys, this means that the keys $\mathsf{pk} = \mathsf{PRerand}(\mathsf{pk}_{T_j,0}, \hat{H}(\mathsf{pk}_{T_j,0}, ch_{T_j}, \cdot, \cdot))$ that tokens use are independent, and the claim follows. To show the claimed domain separation, we can not rely on entropy of the values cid as we did for the variant kdfPA. Indeed, variant bip32PA derives cid deterministically from the server identity id_S. However, as we consider weak unlinkability, this argument is not necessary. Namely, for this specific scheme, credential separation implies that the adversary never uses the same server identity in both Challenge and Left, Right. Then, the domain separation follows from the fact that Left, Right and Challenge access the oracles G_1 and G_2 only on inputs that contain the server identity. □

7.3 Impersonation Security

As showing impersonation security with global revocation is technically the most interesting part of our analysis, we dedicate this entire section to it. We will give an intuitive overview of our analysis, and postpone the formal definitions and proofs to Appendix C.

Unforgeability of Key-Prefixed ECDSA. The reader may wonder why we can not follow the proof idea of variant kdfPA and reduce to the $\mathbf{euf\text{-}cma}$ security of plain ECDSA. To understand this, we have to recall that in the security experiment, the adversary gets the revocation keys $\mathsf{rk}_T = (\mathsf{pk}_{T,0}, ch_T)$ for all tokens T, and a token computes the keys pk that it uses via $\mathsf{pk} := \mathsf{PRerand}(\mathsf{pk}_{T,0}, \hat{H}(\mathsf{pk}_{T,0}, ch, \mathsf{id}_S))$. Therefore, the adversary knows a non-trivial correlation between these public keys, which may allow to run some related key attack. To rule out this attack, the security notion of unforgeability under honestly rerandomized keys ($\mathbf{uf\text{-}hrk}1$) has been introduced [DFL19, DEF+21]. This notion is similar to standard $\mathbf{euf\text{-}cma}$, but in addition, the adversary gets access to an oracle RandO that outputs uniform randomness ρ. An adversary can also ask the signing oracle to sign a message m using rerandomized keys $\mathsf{sk}_\rho := \mathsf{SRerand}(\mathsf{sk}, \rho)$. In the end, a forgery is also allowed to be for a rerandomization of the public key that the adversary obtained. We have the additional restriction that the signing oracle can be queried at most once per pair (m, ρ). Das et al. [DEF+21] showed that the key-prefixed variant of ECDSA satisfies this security notion.

Lemma 7 (informal, [DEF+21])**.** *Let $H : \{0,1\}^* \to \mathbb{Z}_p$ be a random oracle and SIG^{kp}_H be the key-prefixed ECDSA signature scheme. Then, under suitable assumption, for each efficient algorithm $\mathcal{A}$, the advantage $\mathbf{Adv}^{\mathcal{A}}_{\mathbf{uf\text{-}hrk}1,\mathsf{SIG}^{kp}_H}$ is negligible.*

We could just use this result and obtain impersonation security for the key-prefixed variant of bip32PA. As we aim to avoid key-prefixing, we need to translate the above result into a result for plain ECDSA.

Translation to Plain ECDSA. Our main insight is that the proof by Das et al. still works if we prefix all signed messages with some index idx, as long as there is a mapping from idx to ρ, meaning that for each idx, we always use the same $\rho = \rho_{idx}$. Then, in our scheme, we can use the hash of the server identifier id_S as idx. This works because in WebAuthn, messages are always prefixed with the hash of id_S. However, to have this mapping also means that we can not use different cid for the same id_S, which explains why we deterministically derive cid from id_S.

We make this approach more formal by introducing a variant $\mathbf{uf\text{-}hrk\text{-}idx}1$ of the above security notion. In this variant, oracle RandO takes as input an index idx and always outputs the same ρ_{idx} for the same input idx. Then, all signed messages are prefixed by idx in the signing oracle. The final forgery also has to be prefixed with the correct index. Then, we show that ECDSA without key-prefixing satisfies this notion.

Lemma 8. *Let $H_0 : \{0,1\}^* \to \mathbb{Z}_p$ and $H_1 : \{0,1\}^* \to \mathbb{Z}_p$ be random oracles. Let SIG_{H_1} be the ECDSA signature scheme and $\mathsf{SIG}^{kp}_{H_0}$ be its key-prefixed variant as above. Let $\mathcal{A}$ be an adversary in the game $\mathbf{uf\text{-}hrk\text{-}idx}1$ for SIG_{H_1}. Assume that $\mathcal{A}$ makes at most Q_{H_1} queries to random oracle H_1, and at most Q_R queries to oracle* RandO. *Then there exists an algorithm $\mathcal{B}$ with the same running time as $\mathcal{A}$ such that*

$$\mathbf{Adv}^{\mathcal{A}}_{\mathbf{uf\text{-}hrk\text{-}idx}1,\mathsf{SIG}_{H_1}} \leq \frac{(Q_R + Q_{H_1})^2}{2^\lambda} + \mathbf{Adv}^{\mathcal{B}}_{\mathbf{uf\text{-}hrk}1,\mathsf{SIG}^{kp}_{H_0}}.$$

The idea of the proof is that the reduction can internally simulate random oracle $H_1(idx, \cdot)$ by random oracle $H_0(\mathsf{pk}_{idx}, \cdot)$, where pk_{idx} is given as $\mathsf{PRerand}(\mathsf{pk}, \rho_{idx})$.

Using this result, we can then show impersonation security with global revocation of bip32PA. This is done by guessing the token T^* for which the adversary will forge a signature, and then embedding the key given by the $\mathbf{uf\text{-}hrk\text{-}idx}1$ experiment as $\mathsf{pk}_{T^*,0}$. To see that we only need to query the signing oracle once per pair (m, ρ), we highlight that we derive the random coins for signing deterministically in bip32PA. We obtain the following statement.

Lemma 9. *Let $\mathcal{A}$ be an adversary in the impersonation with global revocation game 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 impersonation with global revocation game can be upper bounded by*

$$\frac{Q_H^2 + |\mathcal{S}|^2}{2^{2\lambda}} + \frac{Q_S^2 + Q_{\hat{H}} \cdot |\mathcal{T}|}{2^\lambda} + |\mathcal{T}| \cdot \mathbf{Adv}^{\mathcal{B}}_{\mathbf{uf\text{-}hrk\text{-}idx}1,\mathsf{SIG}_{H'}}.$$

7.4 Evaluation

To show the practicality of bip32PA, we created a prototype implementation and evaluated it on the FIDO2 compliant SoloKey token. The SoloKey firmware uses the secp256r1 curve, is open source and can be uploaded to a special version called Solo Hacker[3]. Additionally, we evaluate the efficiency of the revocation process introduced for bip32PA on a standard Macbook Pro with an Intel i7 processor @2,3 GHz with 4 cores and 16 GB of RAM. We show that revoking even 2^{20} (around one million) tokens takes no more than 3 minutes on this personal computer. It is worth noting that the revocation process is highly parallelizable, and executing it on a server-based platform with multiple cores will yield significantly better execution times. We make the source code for our prototype implementation publicly available[4].

Token Implementation. We evaluate the on-token execution time for registration and authentication of bip32PA and compare it to the execution time of the kdfPA variant that is implemented in the original firmware of SoloKey. The results are presented in Table 2. Our prototype implementation of bip32PA is only around $1.2\times$ as slow as the original firmware implementation of kdfPA. The difference in execution time is around 40 ms in the case of authentication, which will only negligibly influence the user experience.

Global Revocation in Practice. We discuss how services can practically implement revocation. Recall that given the revocation key rk (computed using Revoke), the relying party with identity id_S can verify using CheckCred if the key corresponds to a credential cred. The way we modeled global revocation, the knowledge of rk only allows

[3] https://solokeys.com/collections/all/products/solo-hacker/

[4] Code available here: https://anonymous.4open.science/r/bip32PA

Scheme	Registration Time [ms]	Authentication Time [ms]
kdfPA	369.42	176.31
bip32PA	436.70	213.40

Table 2: Execution time for bip32PA and kdfPA averaged across 100 runs. For all measurements, we instrument the token to not wait for user touch. The standard deviation for all measurements is less than 6ms.

an adversary to link credentials created by the same token and does not allow for impersonation attacks. Thus, users can generate the revocation key during a setup phase and securely extract it from the token to store it, e.g., on a piece of paper as a QR code. A straightaway way to implement the revocation mechanism is to use an approach similar to certificate revocation lists (CRLs) [CSF+08]. The relying party downloads a size B blocklist with revocation keys of revoked tokens and uses the CheckCred algorithm against an internal database containing N unique credentials cred. The complexity of this algorithm is $B \cdot N$, making it somewhat inefficient for services with many users (big N). Like in the case of CRLs, the relying party can download the list via delta updates and only periodically run this check. However, in the case of freshly registered tokens, the relying party must check the revocation status for all elements on the blocklist. For a more in-depth analysis of how to efficiently implement the update, storage, and operation of such a revocation list, we refer to the literature on CRLs.

We will now look at how to optimize the revocation process in the case of bip32PA. The revocation key contains (pk_0, ch) and the CheckCred algorithm computes the public key $\mathsf{pk}' := \mathsf{PRerand}(\mathsf{pk}_0, \hat{H}(\mathsf{pk}_0, ch, \mathsf{id}_S))$ and compares pk' against the credential, which in this case is also a public key pk. In other words, the relying party recomputes the public key the revoked token used for registration, and checks it against the provided public key. There are two observations here that will provide an improvement to the direct approach to revocation described above. Firstly, for each entry on the blocklist, the corresponding public key pk' can be computed once and used against the whole database of the relying party. Secondly, since we compare public keys, we can use a binary search instead of comparing the recomputed public key pk' with every element in the database. The mechanism will look as follows, assuming a presorted database. For each revocation key $\mathsf{rk} := (\mathsf{pk}_0, ch)$ on the blocklist, first recompute $\mathsf{pk}' := \mathsf{PRerand}(\mathsf{pk}_0, \hat{H}(\mathsf{pk}_0, ch, \mathsf{id}_S))$ using the identity id_S and then run a binary search on internal database. The complexity of this mechanism is B times the cost of PRerand and $O(B \cdot \log(N))$ times the cost of comparing public keys (done via BigInteger comparison). It is worth noting that the relying party can store the precomputed public keys and use them in case of new registrations. It can then check the status of a newly registered token with $O(\log(B))$ public key comparisons.

To evaluate its efficiency, we created a prototype implementation of the revocation mechanism described above. We designed a basic function that simulates just one revocation key checking. It generates the public key pk' by reusing the PRerand function from our bip32PA token implementation. Then it simulates a binary search by performing $\log(N)$ comparisons of the public key pk' with a random public key. During each test, we execute this simple function B times. It is easy to see that this process is highly parallelizable, and we used this in our prototype implementation. We divided the work across $T = 16$ threads. Further increasing this number did not significantly improve the mechanism's efficiency on our test platform. However, executing this mechanism on a server-oriented platform can make use of this parallelization even more. We executed and computed the average of 100 test for parameters $N \in \{2^{30}, 2^{32}, 2^{34}, 2^{36}, 2^{38}, 2^{40}\}$ and $B \in \{2^{18}, 2^{20}, 2^{22}, 2^{24}\}$. It is worth noting that the binary search only constituted around 1 second in total for all cases of N. Thus, we can practically use our revocation mechanism with even bigger database sizes. Due to this reason, in Table 3 we present the results only for $N = 2^{40}$. Checking the revocation status of around 1 million ($\sim 2^{20}$) revocation keys takes only about 3 minutes on this personal computer platform. The relying party can check smaller lists of size $2^{18} = 262144$ in less than a minute. It is easy to see that this revocation mechanism is practical. As we mentioned above, blocklists can be provided periodically, and we can realistically assume that they will probably not exceed 2^{20} entries.

Size B	2^{18}	2^{20}	2^{22}	2^{24}
Time [s]	44	175	699	2761

Table 3: Execution time for bip32PA revocation averaged across 100 runs. We assume the size of the relying party's database to be $N = 2^{40}$ and the blocklist size to be B. The standard deviation for all measurements is less than 5s.

8 Conclusion

We analyzed the WebAuthn protocol in FIDO2 with a focus on its real-world use cases and adapted the existing security models accordingly. We showed that privacy (unlinkability) of the protocol is not immediately guaranteed by the specification if keys are stored externally as in common implementations. 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. We also introduced the notion of global key revocation and gave the first formalization of this property. Finally, we have shown that BIP32 key derivation can be used to obtain an efficient token implementation that supports global key revocation and is fully compatible with existing server implementations.

Acknowledgement. We thank the authors of [BBC+25] to point out a minor issue in our original definition of unlinkability, which has been fixed on March 13, 2025.

References

[AKB19] Aftab Alam, Katharina Krombholz, and Sven Bugiel. Poster: Let history not repeat itself (this time) - tackling WebAuthn developer issues early on. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2669–2671. ACM Press, November 2019. (Cited on Page 4.)

[BBC+25] Manuel Barbosa, Alexandra Boldyreva, Shan Chen, Kaishuo Cheng, and Luís Esquível. Revisiting the security and privacy of FIDO2. Cryptology ePrint Archive, Paper 2025/459, 2025. (Cited on Page 20.)

[BBCW21] Manuel Barbosa, Alexandra Boldyreva, Shan Chen, and Bogdan Warinschi. Provable security analysis of FIDO2. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 125–156, Virtual Event, August 2021. Springer, Heidelberg. (Cited on Page 1, 3, 4, 6, 8, 10.)

[BCZ22] Nina Bindel, Cas Cremers, and Mang Zhao. FIDO2, CTAP 2.1, and WebAuthn 2: Provable security and post-quantum instantiation. Cryptology ePrint Archive, Report 2022/1029, 2022. https://eprint.iacr.org/2022/1029. (Cited on Page 2, 3.)

[BF17] Jacqueline Brendel and Marc Fischlin. Zero round-trip time for the extended access control protocol. In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *ESORICS 2017, Part I*, volume 10492 of *LNCS*, pages 297–314. Springer, Heidelberg, September 2017. (Cited on Page 5.)

[BMW03] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 614–629. Springer, Heidelberg, May 2003. (Cited on Page 5.)

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, Heidelberg, May 2000. (Cited on Page 9.)

[BS04] Dan Boneh and Hovav Shacham. Group signatures with verifier-local revocation. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick McDaniel, editors, *ACM CCS 2004*, pages 168–177. ACM Press, October 2004. (Cited on Page 4.)

[CB19] Dhiman Chakraborty and Sven Bugiel. simFIDO: FIDO2 user authentication with simTPM. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2569–2571. ACM Press, November 2019. (Cited on Page 4.)

[CDH16] Jan Camenisch, Manu Drijvers, and Jan Hajny. Scalable revocation scheme for anonymous credentials based on n-times unlinkable proofs. In Edgar R. Weippl, Stefan Katzenbeisser, and Sabrina De Capitani di Vimercati, editors, *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society, WPES@CCS 2016, Vienna, Austria, October 24 - 28, 2016*, pages 123–133. ACM, 2016. (Cited on Page 4.)

[CL02] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 61–76. Springer, Heidelberg, August 2002. (Cited on Page 4.)

[CSF+08] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, RFC Editor, May 2008. (Cited on Page 19.)

[Cv91] David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 257–265. Springer, Heidelberg, April 1991. (Cited on Page 5.)

[DCM+19] Emma Dauterman, Henry Corrigan-Gibbs, David Mazières, Dan Boneh, and Dominic Rizzo. True2F: Backdoor-resistant authentication tokens. In *2019 IEEE Symposium on Security and Privacy*, pages 398–416. IEEE Computer Society Press, May 2019. (Cited on Page 4.)

[DEF+21] Poulami Das, Andreas Erwig, Sebastian Faust, Julian Loss, and Siavash Riahi. The exact security of BIP32 wallets. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 1020–1042. ACM, 2021. (Cited on Page 3, 15, 17.)

[DF11] Özgür Dagdelen and Marc Fischlin. Security analysis of the extended access control protocol for machine readable travel documents. In Mike Burmester, Gene Tsudik, Spyros S. Magliveras, and Ivana Ilic, editors, *ISC 2010*, volume 6531 of *LNCS*, pages 54–68. Springer, Heidelberg, October 2011. (Cited on Page 5.)

[DFL19] Poulami Das, Sebastian Faust, and Julian Loss. A formal treatment of deterministic wallets. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 651–668. ACM Press, November 2019. (Cited on Page 5, 17, 23.)

[DvW92] Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2(2):107–125, June 1992. (Cited on Page 5.)

[FGK+20] Nick Frymann, Daniel Gardham, Franziskus Kiefer, Emil Lundberg, Mark Manulis, and Dain Nilsson. Asynchronous remote key generation: An analysis of yubico's proposal for W3C WebAuthn. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 939–954. ACM Press, November 2020. (Cited on Page 4.)

[fid19] 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]. (Cited on Page 1.)

[FKM+16] Nils Fleischhacker, Johannes Krupp, Giulio Malavolta, Jonas Schneider, Dominique Schröder, and Mark Simkin. Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part I*, volume 9614 of *LNCS*, pages 301–330. Springer, Heidelberg, March 2016. (Cited on Page 5, 23.)

[GH18] Iness Ben Guirat and Harry Halpin. 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, New York, NY, USA, 2018. Association for Computing Machinery. (Cited on Page 2, 4.)

[Jon03] Jakob Jonsson. On the security of CTR + CBC-MAC. In Kaisa Nyberg and Howard M. Heys, editors, *SAC 2002*, volume 2595 of *LNCS*, pages 76–93. Springer, Heidelberg, August 2003. (Cited on Page 24.)

[KHWK22] Michal Kepkowski, Lucjan Hanzlik, Ian Wood, and Mohamed Ali Kâafar. How not to handle keys: Timing attacks on FIDO authenticator privacy. *CoRR*, abs/2205.08071, 2022. (Cited on Page 2, 4, 10.)

[LLM07] Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *ProvSec 2007*, volume 4784 of *LNCS*, pages 1–16. Springer, Heidelberg, November 2007. (Cited on Page 5.)

[LMQ+03] Laurie Law, Alfred Menezes, Minghua Qu, Jerome A. Solinas, and Scott A. Vanstone. An efficient protocol for authenticated key agreement. *Des. Codes Cryptogr.*, 28(2):119–134, 2003. (Cited on Page 5.)

[LSN+20] Sanam Ghorbani Lyastani, Michael Schilling, Michaela Neumayr, Michael Backes, and Sven Bugiel. Is FIDO2 the kingslayer of user authentication? A comparative usability study of FIDO2 passwordless authentication. In *2020 IEEE Symposium on Security and Privacy*, pages 268–285. IEEE Computer Society Press, May 2020. (Cited on Page 2, 4.)

[RBBK01] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: A block-cipher mode of operation for efficient authenticated encryption. In Michael K. Reiter and Pierangela Samarati, editors, *ACM CCS 2001*, pages 196–205. ACM Press, November 2001. (Cited on Page 24.)

[RS06] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 373–390. Springer, Heidelberg, May / June 2006. (Cited on Page 5, 24.)

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, January 1991. (Cited on Page 5.)

[TFS04] Isamu Teranishi, Jun Furukawa, and Kazue Sako. k-Times anonymous authentication (extended abstract). In Pil Joong Lee, editor, *ASIACRYPT 2004*, volume 3329 of *LNCS*, pages 308–322. Springer, Heidelberg, December 2004. (Cited on Page 5.)

[web20] Web Authentication: An API for accessing Public Key Credentials. https://www.w3.org/TR/webauthn/, 2020. [Online; accessed 14-January-2022]. (Cited on Page 1.)

[Wik18] Bitcoin Wiki. Bip32 proposal, 2018. (Cited on Page 15.)

[Yub20] Yubico. Yubikey U2F Key Generation, 2020. [Online; accessed 14-January-2022]. (Cited on Page 2, 5.)

Appendix

A Preliminaries

In this section we fix some notation and provide necessary cryptographic background.

Notation. We denote by $z \leftarrow \mathcal{A}(x)$ the execution of algorithm $\mathcal{A}$ on input x and with output z. If we want to make the random coins ρ of algorithm $\mathcal{A}$ explicit, we write $z := \mathcal{A}(x;\rho)$ instead. 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 $\{1,\ldots,n\}$. By λ, we denote the security parameter. Throughout the paper, we assume public parameters par are given implicitly to all algorithms.

Signatures. We recall the standard notion of digital signatures. We also introduce rerandomizable signature schemes [FKM+16, DFL19] and the corresponding security notion.

Definition 11 (Digital Signature Scheme)**.** *A* digital signature scheme *is a tuple of algorithms* $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ *with the following properties.*

- *The randomized* key generation algorithm Gen *takes as input parameters* par*. It outputs a secret key and public key*$(\mathsf{sk}, \mathsf{pk})$.
- *The randomized* signing algorithm Sig *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* $(\mathsf{sk}, \mathsf{pk}) \in \mathsf{Gen}(\mathsf{par})$ *and all messages* $m \in \{0,1\}^*$ *we have*

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

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 12 (Unforgeability under Chosen Message Attacks)**.** *Let* $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ *be a digitalsignature scheme and consider the experiment* $\mathbf{euf\text{-}cma}^{\mathcal{A}}_{\mathsf{SIG}}$ *defined as follows:*

- **Setup.** *The experiment generates* $(\mathsf{sk}, \mathsf{pk})$ *via* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(\mathsf{par})$*. It runs the adversary* $\mathcal{A}$ *on input* pk*.*
- **Online Phase.** *In this phase,* $\mathcal{A}$ *is given access to oracle* SignO*, which takes as input a message* m *and returns the signature* $\sigma \leftarrow \mathsf{Sig}(\mathsf{sk}, m)$*.*
- **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* SignO*. Otherwise, it returns* 0*.*

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

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

Definition 13 (Rerandomizable Signature [FKM+16, DFL19])**.** *A* rerandomizable signature scheme *is a tuple of algorithms* $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{SRerand}, \mathsf{PRerand}, \mathsf{Ver})$*, where* $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ *is a digital signature scheme, and algorithms* $\mathsf{SRerand}, \mathsf{PRerand}$ *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 $(\mathsf{sk}, \mathsf{pk}) \in \mathsf{Gen}(\mathsf{par})$*, key pairs* $(\mathsf{sk}_\rho, \mathsf{pk}_\rho)$ *generated as* $\rho \xleftarrow{\$} \{0,1\}^\lambda, \mathsf{sk}_\rho := \mathsf{SRerand}(\mathsf{sk}, \rho)$*, and* $\mathsf{pk}_\rho := \mathsf{PRerand}(\mathsf{pk}, \rho)$ *are identically distributed to key pairs generated via* $(\mathsf{sk}', \mathsf{pk}') \leftarrow \mathsf{Gen}(\mathsf{par})$.

Symmetric Primitives. Here, we recall the definition of symmetric key encryption that is both authenticated and anonymous. For the definition of security, we follow [RS06]. For simplicity of exposition, we choose to leave nonces implicit, leading to a potentially weaker notion. We note that this notion is satisfied using the OCB mode of operation [RBBK01] or the CCM mode of operation [Jon03].

Definition 14 (Symmetric Key Encryption Scheme)**.** *A* symmetric key encryption scheme *(SKE) with length function* $\nu \colon \mathbb{N} \to \mathbb{N}$ *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* par*. 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* $c \in \{0,1\}^{\nu(|m|)}$.
- *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 15 (Authenticated Anonymous Security for SKE)**.** *Let* $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a symmetric key encryption scheme with length function* ν*. 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* $\mathsf{sk} \leftarrow \mathsf{Gen}(\mathsf{par})$ *and initializes a map* $L[\cdot]$.
- **Online Phase.** *The adversary* $\mathcal{A}$ *is run on input* par *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* $c \xleftarrow{\$} \{0,1\}^{\nu(|m|)}$ *uniformly at random, define* $L[c] := m$ *and return* c.
  - $\mathsf{DecO}_1(c)$*: If* $L[c] \neq \perp$*, return* $L[c]$*. Else, 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*

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

Definition 16 (Pseudorandom Function)**.** *Let* $\ell_1 = \ell_1(\lambda), \ell_2 = \ell_2(\lambda) \in \mathbb{N}$*. Consider a efficiently computable function* $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^{\ell_1} \to \{0,1\}^{\ell_2}$*, an algorithm* $\mathcal{A}$*, a bit* $b \in \{0,1\}$*, and the following experiment* $\mathbf{prf}^{\mathcal{A}}_{\mathsf{PRF},b}$*:*

- **Setup.** *The experiment samples* $K \xleftarrow{\$} \{0,1\}^\lambda$ *and initializes a map* $F[\cdot]$.
- **Online Phase.** *The adversary* $\mathcal{A}$ *is run on input* par *with oracle access to oracle* EvalO_b*, which is defined as follows.*
  - $\mathsf{EvalO}_0(x \in \{0,1\}^{\ell_1})$*: If* $F[x] \neq \perp$*, return* $F[x]$*. Else, sample* $F[x] \xleftarrow{\$} \{0,1\}^{\ell_2}$ *and return* $F[x]$.
  - $\mathsf{EvalO}_1(x \in \{0,1\}^{\ell_1})$*: Return* $\mathsf{PRF}(K, x)$.
- **Output Phase.** *The adversary outputs a bit* b'*. The experiment outputs* b'.

We define the advantage of $\mathcal{A}$ *against the pseudorandomness of* PRF *as*

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

B Omitted Proofs for Existing Implementations

B.1 Omitted Proofs For Key Derivation Function

Proof of Lemma 1. 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 impersonation 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 and a random oracle H. By definition, we have

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

Game $\mathbf{G}_1$: We rule out a 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

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

Game $\mathbf{G}_2$: In this game, we introduce 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

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

Game $\mathbf{G}_3$: In this game, we introduce yet another bad event, for which we let the game abort on its occurrence. Namely, consider a query of the form Challenge$(\pi_T^{i,0}, \cdot)$. Recall that in such a query, a value cid is sampled uniformly at random. Now, the game aborts if this cid has been output or input of a previous query of the form Challenge$(\pi_T^{i',j}, \cdot)$. Clearly, we have

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

Game $\mathbf{G}_4$: In this game, we change the way the secret signing keys sk used in queries of the form Challenge$(\pi_T^{i,j}, \mathsf{id}_S, \mathsf{cid}, \cdot)$ are derived. Namely, before they were derived as $\mathsf{sk} := \mathsf{PRF}(\mathsf{msk}_T, (\mathsf{cid}, \mathsf{id}_S)$. Now, the game samples them at random and stores an initially empty map $K_T[\cdot]$ (for each token $T \in \mathcal{T}$) to ensure consistency. Concretely, the game first checks if $K_T[\mathsf{cid}, \mathsf{id}_S]$ is already defined. If it is not, it samples a secret key sk uniformly at random, and sets $K_T[\mathsf{cid}, \mathsf{id}_S] := \mathsf{sk}$. In any case, it uses $\mathsf{sk} := K_T[\mathsf{cid}, \mathsf{id}_S]$ to continue. We can use the pseudorandomness of PRF to bound the distinguishing advantage between $\mathbf{G}_3$ and $\mathbf{G}_4$. Namely, we use a hybrid argument over the $|\mathcal{T}|$ tokens, and in each hybrid step, we introduce the map $K_T[\cdot]$ as above for the next token T. We can show that two hybrids are close using the pseudorandomness of PRF, via a straight-forward reduction $\mathcal{B}'$, and get

$$|\mathbf{Adv}_3 - \mathbf{Adv}_4| \leq |\mathcal{T}| \cdot \mathbf{Adv}^{\mathcal{B}'}_{\mathbf{prf},\mathsf{PRF}}.$$

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 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 $\mathbf{euf\text{-}cma}$ security of SIG. The reduction is as follows.

- $\mathcal{B}$ gets as input a public key pk^*. It gets access to a signing oracle SignO.

- $\mathcal{B}$ 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 Challenge$(\pi_T^{i,j}, \cdot)$ for $j > 0$, then $\mathcal{B}$ aborts.
  - If this query is a registration query, i.e. it is of the form Challenge$(\pi_T^{i,0}, \cdot)$, then it sets pk := pk*, and continues the simulation using this key. Thereby, it obtains the signature σ from its signing oracle SignO. Let cid be the random credential identifier sampled within this simulation. Later, whenever the adversary queries oracle Challenge for authentications for this cid, $\mathcal{B}$ uses its signing oracle SignO 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 $\mathbf{G}_1$ and $\mathbf{G}_2$, if $\mathcal{A}$ wins game $\mathbf{G}_4$, we know that the signing oracle SignO 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

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

□

Proof of Lemma 3. 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 1.

Game $\mathbf{G}_0$: This game is the real medium 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 Challenge$(\pi_{T_b}^{i_b,j_b}, \mathsf{id}_{S_L}, \cdot, \cdot)$ and Challenge$(\pi_{T_{1-b}}^{i_{1-b},j_{1-b}}, \mathsf{id}_{S_R}, \cdot, \cdot)$, respectively. By definition, we have

$$\mathbf{Adv}^{\mathcal{A}}_{\mathbf{mUnl},\mathsf{kdfPA}} = \left|\mathsf{pr}_0 - \frac{1}{2}\right|.$$

Game $\mathbf{G}_1$: This game is as $\mathbf{G}_0$, but we introduce a bad event and let the game abort if it occurs. Consider the lists $\mathcal{L}^r_{lr}$ and $\mathcal{L}^r_{ch}$ as in the definition of the unlinkability game. Game $\mathbf{G}_1$ aborts if there is a collision, i.e. $\mathcal{L}^r_{lr} \cap \mathcal{L}^r_{ch} \neq \emptyset$. Recall that the values cid is sampled uniformly at random, and $|\mathcal{L}^r_{lr}| \leq 2$. Therefore, we have

$$|\mathsf{pr}_0 - \mathsf{pr}_1| \leq \frac{2Q_C}{2^\lambda}.$$

Game $\mathbf{G}_2$: This game is as $\mathbf{G}_1$, but we change the way the secret signing keys sk used in queries of the form Challenge$(\pi_T^{i,j}, \mathsf{id}_S, \mathsf{cid}, \cdot)$ or queries to Left and Right are derived. This is similar to the proof of impersonation security. Recall that before they were derived as sk := PRF$(\mathsf{msk}_T, (\mathsf{cid}, \mathsf{id}_S)$ for each token $T \in \mathcal{T}$. Now, we introduce an initially empty map $K_T[\cdot]$ for each token $T \in \mathcal{T}$. Then, whenever $\mathbf{G}_1$ would compute sk as above, game $\mathbf{G}_2$ first checks if $K_T[\mathsf{cid}, \mathsf{id}_S]$ is already defined. If it is not, it samples a secret key sk uniformly at random, and sets $K_T[\mathsf{cid}, \mathsf{id}_S] := \mathsf{sk}$. In any case, it uses sk := $K_T[\mathsf{cid}, \mathsf{id}_S]$ to continue. As in the proof of impersonation

security, we can use the pseudorandomness of PRF in $|\mathcal{T}|$ hybrid steps to bound the distinguishing advantage between $\mathbf{G}_1$ and $\mathbf{G}_2$. To subsequent hybrids are shown to be close via a straight-forward reduction $\mathcal{B}$, and we get

$$|\mathsf{pr}_1 - \mathsf{pr}_2| \leq |\mathcal{T}| \cdot \mathbf{Adv}^{\mathcal{B}}_{\mathbf{prf},\mathsf{PRF}}.$$

Game $\mathbf{G}_3$: This game is as $\mathbf{G}_2$, but we split the map K_{T_b} into two maps. To recall, T_b is the challenge token used in oracle Left. In game $\mathbf{G}_3$, instead of having one map K_{T_b} that is used in Challenge$(\pi^{i,j}_{T_b}, \cdot)$ and Left, we now only use K_{T_b} in Challenge$(\pi^{i,j}_{T_b}, \cdot)$, and a separate independent map $K_L[\cdot]$ in oracle Left. It follows from credential separation and the bad event that we ruled out in $\mathbf{G}_1$ that this is only a conceptual change. Namely, the change can only be noticed, if the game accesses both $K_{T_b}[\mathsf{cid}, \mathsf{id}_S]$ and $K_L[\mathsf{cid}, \mathsf{id}_S]$ for some $(\mathsf{cid}, \mathsf{id}_S)$. By definition of oracle Left, this means that $S = S_L$, and therefore credential separation applies. We get

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

Game $\mathbf{G}_4$: This game is as game $\mathbf{G}_3$, with a similar change as the previous one but for oracle Right and token T_{1-b}. Namely, we split the map $K_{T_{1-b}}$ into two maps. The map $K_{T_{1-b}}$ is used in Challenge$(\pi^{i,j}_{T_{1-b}}, \cdot)$ and a map $K_R[\cdot]$ is used in oracle Right. Similarly as above, we get

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

We highlight that the steps from $\mathbf{G}_2$ to $\mathbf{G}_4$ would not be correct if we tried to prove strong unlinkability. This is because in this case, $\mathcal{A}$ would be allowed to come up with some fresh cid and submit it to oracle Left and Challenge, which react consistently in $\mathbf{G}_2$ and inconsistently in $\mathbf{G}_3$.

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. □

B.2 Omitted Proofs For Key Wrapping

Proof of Lemma 2. 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 impersonation game. At the beginning of this game, every token $T \in \mathcal{T}$ is initialized with a master secret key $\mathsf{msk}_T \leftarrow \mathsf{Gen}(\mathsf{par})$. Then, the adversary gets access to oracles Start, Challenge, Complete. By definition, we have

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

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 Challenge$(\pi^{i,0}_T, \mathsf{id}_S, \cdot)$ (i.e. in registration) and to decrypt such cid in queries of the form Challenge$(\pi^{i,j}_T, \mathsf{id}_S, \mathsf{cid}, \cdot)$, $j > 0$ (i.e. in authentication). In game $\mathbf{G}_2$, we instead hold an initially empty map $L[\cdot,\cdot]$. In each registration query Challenge$(\pi^{i,0}_T, \mathsf{id}_S, \cdot)$ the value cid is now sampled uniformly at random from $\{0,1\}^{\nu^*}$, and an entry $L[T, \mathsf{cid}] := (\mathsf{id}_S, \mathsf{sk})$ is added. In each authentication query Challenge$(\pi^{i,j}_T, \mathsf{id}_S, \mathsf{cid}, \cdot), j > 0$, 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 1 we can build a reduction $\mathcal{C}$ from the $\mathbf{euf\text{-}cma}$ security of SIG to bound $\mathbf{Adv}_3$. At a high level, the reduction guesses in which query of the form Challenge$(\pi_T^{i,0}, \cdot)$ the adversary obtained the public key, for which it forges a signature. We have

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

□

Proof of Lemma 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 strong unlinkability game. In this game, a key $\mathsf{sk}_T \leftarrow \mathsf{Gen}(\mathsf{par})$ 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 Challenge$(\pi_{T_b}^{i_b,j_b}, \mathsf{id}_{S_L}, \cdot, \cdot)$ and Challenge$(\pi_{T_{1-b}}^{i_{1-b},j_{1-b}}, \mathsf{id}_{S_R}, \cdot, \cdot)$, respectively. By definition, we have

$$\mathbf{Adv}^{\mathcal{A}}_{\mathbf{sUnl},\mathsf{kwrPA}} = \left|\mathsf{pr}_0 - \frac{1}{2}\right|.$$

Game $\mathbf{G}_1$: We change game $\mathbf{G}_0$ in the following way. In 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. In $\mathbf{G}_1$, whenever $\mathcal{A}$ starts a registration with token $T_r, r \in \{0,1\}$ via oracles Challenge or Left, Right, a ciphertext cid is created as $\mathsf{cid} := \mathsf{Enc}(\mathsf{msk}_{T_r}, (\mathsf{id}_S, \mathsf{sk}))$. Furthermore, when $\mathcal{A}$ starts an authentication interaction with token $T_r, r \in \{0,1\}$ 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 $\{0,1\}^{\nu^*}$. 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 $\mathbf{G}_3$: Similar to 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 the same cid is given out in registration twice (once in Left, Right, and once in Challenge), or 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. The former only occurs with probability at most $2Q_C/|\{0,1\}^{\nu^*}|$, and the latter is forbidden due to (strong) credential separation. It follows that

$$|\mathsf{pr}_2 - \mathsf{pr}_3| \leq \frac{2Q_C}{2^{\nu^*}}.$$

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 $\mathsf{pr}_4 = 1/2$. □

C Omitted Definitions and Proofs For BIP32

Lemma 10. *Let* $\mathsf{SIG}_H = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{SRerand}, \mathsf{PRerand}, \mathsf{Ver})$ *be 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* par *and obtain a string* $id \in \{0,1\}^*$.
2. *Generate a key pair* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(\mathsf{par})$ *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 5. 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 $\mathsf{msk}_T = (\mathsf{sk}_{T,0}, \mathsf{pk}_{T,0}, ch_T, seed_T)$ is generated for each token $T \in \mathcal{T}$. Then, the adversary gets access to oracles Start, Challenge, Complete and outputs tuples (T^*, i_{T^*}) and (S^*, i_{S^*}). Afterwards, it gets the credential $\mathsf{cred}^* = \mathsf{pk}$ and outputs a revocation key rk^*. The game outputs 1 if and only if the oracles $\pi_{T^*}^{i_{T^*},0}$ and $\pi_{S^*}^{i_{S^*},0}$ are partnered and $\mathsf{CheckCred}(\mathsf{id}_{S^*}, \mathsf{cred}^*, \mathsf{rk}^*)$. By definition, we have

$$\mathbf{Adv}_1 = \mathbf{Adv}^{\mathcal{A}}_{\mathbf{rev\text{-}sound},\mathsf{bip32PA}}.$$

Game $\mathbf{G}_1$: This game is as $\mathbf{G}_0$, but we add a 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 for some token $T \in \mathcal{T}$. 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 and all tokens $T \in \mathcal{T}$ we get

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

Game $\mathbf{G}_2$: This game is as $\mathbf{G}_1$, but we rule out another bad event. Namely, the game aborts if there is a collision for random oracle $\hat{H}$, i.e. there are $x \neq x'$ such that $\hat{H}(x) = \hat{H}(x')$. Clearly, we have

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

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. 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,

$$|\mathbf{Adv}_2 - \mathbf{Adv}_3| \leq \frac{1}{2^\lambda}.$$

Game $\mathbf{G}_4$: This game is as game $\mathbf{G}_3$, but it samples an index $I \xleftarrow{\$} [Q_C]$ uniformly at random in the beginning. Then, it aborts in the end, if the Ith query to oracle Challenge is not the first registration query for token T^* and server identity id_{S^*}. More formally, it is not the first query of the form $\mathsf{Challenge}(\pi_{T^*}^{\cdot,0}, \mathsf{id}_{S^*}, \cdot, \cdot)$. By definition of the game, such a query must exist if the adversary wins the game. Intuitively, the game guesses which query to oracle Challenge defines credential that is attacked by the adversary. As $\mathcal{A}$'s view is independent of I until the potential abort, we have

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

Game $\mathbf{G}_5$: This game is as $\mathbf{G}_4$, but we change the execution of the Ith query to oracle Challenge. Recall that in previous games, during that registration, a key pair $(\mathsf{sk}, \mathsf{pk})$ is generated using $\rho := \hat{H}(\mathsf{pk}_{T^*,0}, ch_{T^*}, \mathsf{id}_{S^*})$ and

$$\mathsf{sk} := \mathsf{SRerand}(\mathsf{sk}_{T^*,0}, \rho), \quad \mathsf{pk} := \mathsf{PRerand}(\mathsf{pk}_{T^*}, \rho).$$

Especially, assuming that $\mathbf{G}_4$ does not abort, this is the first query for T^* and id_{S^*}, and due to the change in $\mathbf{G}_1$, the value ρ is uniformly random at this point. In game $\mathbf{G}_5$, the game samples $(\mathsf{sk}, \mathsf{pk})$ as a fresh key pair via $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(\mathsf{par})$ and programs

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

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

$$\mathbf{Adv}_4 = \mathbf{Adv}_5.$$

Finally, we can bound the advantage $\mathbf{Adv}_5$ of $\mathcal{A}$ in game $\mathbf{G}_5$ using a reduction from the game in Lemma 10. The reduction gets as input parameters par and gets oracle access to an oracle $\tilde{H}$. It sets up the game for $\mathcal{A}$ as in $\mathbf{G}_5$, while simulating oracle $\hat{H}$ by forwarding queries to $\tilde{H}$. Then, in the Ith query to oracle Challenge, the reduction outputs $id := \mathsf{id}_{S^*}$ to its game. It obtains $(\mathsf{sk}, \mathsf{pk})$ from its game and programs

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

as in $\mathbf{G}_5$. 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 $\mathbf{G}_5$ for $\mathcal{A}$. Moreover, if the bad event defined in $\mathbf{G}_3$ 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 10 implies that $\mathbf{Adv}_5 \leq Q_{\hat{H}}/2^\lambda$ and the statement follows. □

Proof of Lemma 6. 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 $\mathsf{msk}_T = (\mathsf{sk}_{T,0}, \mathsf{pk}_{T,0}, ch_T, seed_T)$ is generated, and a revocation key $\mathsf{rk}_T \leftarrow \mathsf{Revoke}(\mathsf{msk}_T)$ is derived for each token $T \in \mathcal{T}$. Adversary $\mathcal{A}$ gets access to oracles Start, Challenge, Complete. Then, it outputs two challenge tokens T_0, T_1

and servers S_L, S_R. Afterwards, it also gets $\{\mathsf{rk}_T\}_{T\in\mathcal{T}\setminus\{T_0,T_1\}}$, and access to oracles Left, Right, which internally run Challenge($\pi_{T_b}^{i_b,j_b}, \mathsf{id}_{S_L}, \cdot, \cdot$) and 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

$$\mathbf{Adv}^{\mathcal{A}}_{\mathbf{wUnl\text{-}GR},\mathsf{bip32PA}} = \left|\mathsf{pr}_0 - \frac{1}{2}\right|.$$

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 $\hat{H}(\mathsf{pk}_{T_j,0}, ch_{T_j}, \cdot, \cdot)$ or $\hat{H}(seed_{T_j}, \cdot)$ for one of the challenge tokens $j \in \{0,1\}$. Note that $\mathcal{A}$ obtains no information about ch_{T_j} and $seed_{T_j}$ throughout the game (except via hash values). Thus, we can use a union bound over the two cases, all $Q_{\hat{H}}$ random oracle queries and the two tokens to obtain

$$|\mathsf{pr}_0 - \mathsf{pr}_1| \leq \frac{4 \cdot Q_{\hat{H}}}{2^\lambda}.$$

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 $ch_T = ch_{T'}$ or $seed_T = seed_{T'}$. As these values are sampled uniformly at random from $\{0,1\}^\lambda$ and independently, we have

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

Game $\mathbf{G}_3$: This game is as $\mathbf{G}_2$, but it aborts if there are $x \neq x'$ such that $H(x) = H(x')$. Clearly, we have

$$|\mathsf{pr}_2 - \mathsf{pr}_3| \leq \frac{Q_H^2}{2^{2\lambda}}.$$

Game $\mathbf{G}_4$: This game is as $\mathbf{G}_3$, but we change the way oracle Left works. First, we recall how oracle Left works in $\mathbf{G}_3$. By definition of the protocol, it returns $\mathsf{cid}_L = \hat{H}(seed_{T_b}, \mathsf{id}_{S_L})$ when it is queried for the first time, i.e. during registration. Later, it always aborts if the value cid that is input is not equal to cid_L. By the definition of the bad event in $\mathbf{G}_1$, the value cid_L is distributed uniformly from $\mathcal{A}$'s perspective when $\mathcal{A}$ queries Left for the first time. Next, recall that on input cid_L, M, oracle Left(cid_L, M) in game $\mathbf{G}_3$ samples a key pair $(\mathsf{sk}, \mathsf{pk})$ as follows: It derives $\rho := \hat{H}(\mathsf{pk}_{T_b,0}, ch_{T_b}, \mathsf{id}_{S_L})$ and sets

$$\mathsf{sk} := \mathsf{SRerand}(\mathsf{sk}_{T_b,0}, \rho), \quad \mathsf{pk} := \mathsf{PRerand}(\mathsf{pk}_{T_b,0}, \rho).$$

Then, it computes a signature on a message m using random coins $coins := \hat{H}(seed_{T_b}, m)$.

We remark that the deterministic derivation of cid from server identities id_S in combination with credential seperation implies that id_{S_L} and id_{S_R} are never submitted to oracle Challenge directly. From this, assuming the game $\mathbf{G}_1$ does not abort, we know value ρ is uniformly distributed for $\mathcal{A}$ here. Also, whenever a message m is signed by oracle Left for the first time, the value $\hat{H}(seed_{T_b}, m)$ is uniformly distributed for $\mathcal{A}$. Especially, due to our remark about credential separation above and non-colliding ch's, the oracle Challenge never queries $\hat{H}(\mathsf{pk}_{T_b,0}, ch_{T_b}, \mathsf{id}_{S_L})$. Note that signed messages m always contain $H(\mathsf{id}_S)$ and we ruled out collisions for H, so oracle Challenge never queries $\hat{H}(seed_{T_b}, m)$.

Oracle Left in $\mathbf{G}_4$ now holds a map L_L and works as follows: It samples a random cid_L, and a fresh key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(\mathsf{par})$. When Left is called for the first time, it returns $\mathsf{cid}_L, \mathsf{pk}$ and a signature as in the protocol. For the following calls Left(cid, M), it aborts if $\mathsf{cid} \neq \mathsf{cid}_L$. Otherwise, it answers them as before, but using that particular key pair and randomness $L_L[m]$ to sign a message m. Whenever $L_L[m]$ is not yet defined, it is sampled uniformly at random. The above arguments show that if the game does not abort, the view of $\mathcal{A}$ does not change. Thus we have

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

Game $\mathbf{G}_5$: This game is as $\mathbf{G}_4$, but we change how oracle Right behaves. Concretely, we apply a similar change using fresh key pairs, a random cid_R and a map L_R as we did for oracle Left in game $\mathbf{G}_3$. With the same arguments, it follows that

$$\mathsf{pr}_4 = \mathsf{pr}_5.$$

Finally, note that oracles Left and Right in game $\mathbf{G}_5$ and thus $\mathcal{A}$'s view are independent of the bit b. Thus, we have $\mathsf{pr}_5 = 1/2$ and the claim follows. □

Definition 17 (Unforgeability under Honestly Rerandomized Keys)**.** *Let* $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{SRerand}, \mathsf{PRerand}, \mathsf{Ver})$ *be a rerandomizable signature scheme and consider the experiment* $\mathbf{uf\text{-}hrk1}^{\mathcal{A}}_{\mathsf{SIG}}$ *defined as follows:*

- **Setup.** *The experiment generates* $(\mathsf{sk}, \mathsf{pk})$ *via* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(\mathsf{par})$. *It runs the adversary* $\mathcal{A}$ *on input* pk.
- **Online Phase.** *In this phase,* $\mathcal{A}$ *is given access to oracles* SignO *and* RandO*:*
  - RandO *takes no inputs and returns* $\rho \xleftarrow{\$} \{0,1\}^\lambda$.
  - SignO *takes as input a message* m *and a string* $\rho \in \{0,1\}^\lambda$. *If* ρ *was not a previous output of* RandO *or the pair* (m, ρ) *has been queried before,* SignO *returns* $\perp$. *Otherwise, it returns the signature* σ *computed via* $\sigma \leftarrow \mathsf{Sig}(\mathsf{SRerand}(\mathsf{sk}, \rho), m)$.
- **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* RandO*, and* (m^*, ρ^*) *was not queried to* SignO. *Otherwise, it returns* 0.

We define the advantage of $\mathcal{A}$ *in* $\mathbf{uf\text{-}hrk1}^{\mathcal{A}}_{\mathsf{SIG}}$ *as*

$$\mathbf{Adv}^{\mathcal{A}}_{\mathbf{uf\text{-}hrk1},\mathsf{SIG}} := \Pr[\mathbf{uf\text{-}hrk1}^{\mathcal{A}}_{\mathsf{SIG}} = 1].$$

Definition 18 (Unforgeability under Honestly Rerandomized Keys With Index)**.** *Let* $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{SRerand}, \mathsf{PRerand}, \mathsf{Ver})$ *be a rerandomizable signature scheme. We consider experiment* $\mathbf{uf\text{-}hrk\text{-}idx1}^{\mathcal{A}}_{\mathsf{SIG}}$*, which is defined as follows:*

- **Setup.** *The experiment generates* $(\mathsf{sk}, \mathsf{pk})$ *via* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(\mathsf{par})$. *It runs the adversary* $\mathcal{A}$ *on input* pk.
- **Online Phase.** *In this phase,* $\mathcal{A}$ *is given access to oracles* SignO *and* RandO*:*
  - RandO *takes as input an index* $idx \in \{0,1\}^{2\lambda}$. *If* ρ_{idx} *is not defined yet, it samples* $\rho_{idx} \xleftarrow{\$} \{0,1\}^\lambda$. *It returns* ρ_{idx}.
  - SignO *takes as input a message* m *and an index* $idx \in \{0,1\}^{2\lambda}$. *If the pair* (m, idx) *has been queried before,* SignO *returns* $\perp$. *If* RandO *has never been queried with input* idx *before, it also returns* $\perp$. *Otherwise, it returns the signature* σ *for message* (idx, m) *computed via* $\sigma \leftarrow \mathsf{Sig}(\mathsf{SRerand}(\mathsf{sk}, \rho_{idx}), (idx, m))$.
- **Output Phase.** *When* $\mathcal{A}$ *returns* (σ^*, idx^*, m^*)*, the experiment returns* 0 *if* RandO *has never been queried with input* idx^* *before. Otherwise, it returns* 1 *if* $\mathsf{Ver}(\mathsf{PRerand}(\mathsf{pk}, \rho_{idx^*}), \sigma^*, (idx^*, m^*)) = 1$*, and* (m^*, idx^*) *was not queried to* SignO. *Otherwise, it returns* 0.

We define the advantage of $\mathcal{A}$ *in* $\mathbf{uf\text{-}hrk\text{-}idx1}^{\mathcal{A}}_{\mathsf{SIG}}$ *as*

$$\mathbf{Adv}^{\mathcal{A}}_{\mathbf{uf\text{-}hrk\text{-}idx1},\mathsf{SIG}} := \Pr[\mathbf{uf\text{-}hrk\text{-}idx1}^{\mathcal{A}}_{\mathsf{SIG}} = 1].$$

Proof of Lemma 8. To prove the claim, we present some changes to the game $\mathbf{G}_0 = \mathbf{uf\text{-}hrk\text{-}idx1}^{\mathcal{A}}_{\mathsf{SIG}_{H_1}}$, leading to a sequence of games $\mathbf{G}_i$, $i \in [4]$. 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 game $\mathbf{uf\text{-}hrk\text{-}idx1}^{\mathcal{A}}_{\mathsf{SIG}_{H_1}}$. Here, $\mathcal{A}$ has access to random oracle H_1. First, a pair $(\mathsf{sk}, \mathsf{pk})$ of keys is generated and pk is given to $\mathcal{A}$. Then, $\mathcal{A}$ has access to oracles SignO and RandO. The game internally keeps track of randomness ρ_{idx} that oracle RandO output on input idx. Finally, $\mathcal{A}$ outputs a forgery (σ^*, idx^*, m^*). By definition, we have

$$\mathbf{Adv}_0 = \mathbf{Adv}^{\mathcal{A}}_{\mathbf{uf\text{-}hrk\text{-}idx1},\mathsf{SIG}_{H_1}}.$$

Game $\mathbf{G}_1$: This game is as $\mathbf{G}_0$, but we slightly change the simulation of oracles RandO and H_1. Namely, when $\mathcal{A}$ queries H_1 on input (idx, m), it now checks if ρ_{idx} is already defined. If it is not defined yet, it samples $\rho_{idx} \xleftarrow{\$} \{0,1\}^\lambda$. This ρ_{idx} will later be used by oracle RandO if $\mathcal{A}$ queries idx. Note that oracle SignO still aborts if RandO has never been queried on input idx before, even if ρ_{idx} is already defined due to random oracle queries. Similarly, the winning condition is unchanged. Therefore, this is only a conceptual change, and we have

$$\mathbf{Adv}_0 = \mathbf{Adv}_1.$$

Game $\mathbf{G}_2$: This game is exactly as $\mathbf{G}_1$, but it aborts if there are two different $idx \neq idx'$ for which $\rho_{idx} = \rho_{idx'}$. As the values ρ_{idx} and $\rho_{idx'}$ are sampled uniformly at random from $\{0,1\}^\lambda$, the probability of such a collision for fixed idx, idx' is $1/2^\lambda$. Using a union bound over all pairs that the adversary can query, we derive an upper bound on the probability of such an abort, leading to

$$|\mathbf{Adv}_1 - \mathbf{Adv}_2| \leq \frac{(Q_R + Q_{H_1})^2}{2^\lambda}.$$

Game $\mathbf{G}_3$: We change how the random oracle H_1 is simulated. Concretely, the game now internally runs a random oracle $H_0 : \{0,1\}^* \to \mathbb{Z}_p$, which is not provided to $\mathcal{A}$. For each query $H_1(idx, m)$, it first makes sure that ρ_{idx} is already defined, as introduced in $\mathbf{G}_1$. Then, it computes $\mathsf{pk}_{idx} := \mathsf{PRerand}(\mathsf{pk}, \rho_{idx})$ and returns $H_0(\mathsf{pk}_{idx}, m)$. Due to the change in the previous game, indices idx are injectively mapped to randomness ρ_{idx}. For the concrete ECDSA randomization algorithm PRerand, the mapping $\rho \mapsto \mathsf{PRerand}(\mathsf{pk}, \rho)$ is also injective. Therefore, idx is injectively mapped to pk_{idx}, which implies that

$$\mathbf{Adv}_2 = \mathbf{Adv}_3.$$

Game $\mathbf{G}_4$: We change how oracle SignO is provided. Recall that in previous games, when $\mathcal{A}$ queries SignO on input m and idx, the oracle either aborts, or returns $\sigma \leftarrow \mathsf{Sig}^{int}(\mathsf{SRerand}(\mathsf{sk}, \rho_{idx}), H_1(idx, m))$. From now on, the oracle runs $\sigma \leftarrow \mathsf{Sig}^{int}(\mathsf{SRerand}(\mathsf{sk}, \rho_{idx}), H_0(\mathsf{pk}_{idx}, m))$ instead, where $\mathsf{pk}_{idx} := \mathsf{PRerand}(\mathsf{pk}, \rho_{idx})$. We claim that this change is only syntactical. Indeed, due to the previous change, we have $H_0(\mathsf{pk}_{idx}, m) = H_1(idx, m)$, which implies that

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

Finally, it is easy to see that we can bound $\mathbf{Adv}_4$ using a reduction $\mathcal{B}$ from the game $\mathbf{uf\text{-}hrk1}_{\mathsf{SIG}^{kp}_{H_0}}$. More precisely, the reduction simulates $\mathbf{G}_4$ as follows:

- $\mathcal{B}$ gets as input a public key pk. It gets access to oracle SignO^{kp}, RandO^{kp} and a random oracle H_0.
- $\mathcal{B}$ forwards public key pk to $\mathcal{A}$. It also provides a random oracle H_1 to $\mathcal{A}$, which is simulated using oracle H_0 as in game $\mathbf{G}_4$.
- $\mathcal{B}$ provides oracles SignO and RandO to $\mathcal{A}$. Whenever, $\mathcal{B}$ needs to sample a new randomness ρ_{idx} (either in RandO or in H_1) it queries RandO^{kp} to do so. To simulate oracle SignO on input m and idx, $\mathcal{B}$ either aborts as in $\mathbf{G}_4$ or submits m and ρ_{idx} to SignO^{kp}.
- When $\mathcal{A}$ returns (σ^*, idx^*, m^*), $\mathcal{B}$ first checks the winning conditions as in $\mathbf{G}_4$. If they hold, then ρ_{idx^*} is defined. The reduction $\mathcal{B}$ submits $(\sigma^*, \rho_{idx^*}, m^*)$ as its forgery to the game $\mathbf{uf\text{-}hrk1}_{\mathsf{SIG}^{kp}_{H_0}}$.

It is easy to see that $\mathcal{B}$ perfectly simulates game $\mathbf{G}_4$ for $\mathcal{A}$. Furthermore, due to $H_0(\mathsf{pk}_{idx}, m) = H_1(idx, m)$ and the injectiveness of the mapping $idx \mapsto \rho_{idx}$, reduction $\mathcal{B}$ wins its game whenever $\mathbf{G}_4$ outputs 1. Therefore, we have

$$\mathbf{Adv}_4 \leq \mathbf{Adv}^{\mathcal{B}}_{\mathbf{uf\text{-}hrk1},\mathsf{SIG}^{kp}_{H_0}}.$$

□

Proof of Lemma 9. 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 impersonation with global revocation game. That is, at the beginning a master secret key $\mathsf{msk}_T = (\mathsf{sk}_{T,0}, \mathsf{pk}_{T,0}, ch_T, seed_T)$ 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

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

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. Recall that in an execution of the server-side oracles Start, the game samples $rs \xleftarrow{\$} \{0,1\}^{\geq\lambda}$. 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. Namely, the game aborts if for some token $T \in \mathcal{T}$, $\mathcal{A}$ queries $\hat{H}(seed_T, \cdot)$. As the values $seed_T$ are sampled uniformly at random from $\{0,1\}^{\lambda}$, a union bound over all tokens and all random oracle queries shows that

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

Game $\mathbf{G}_4$: In this game, we add another abort. Namely, the game aborts if there exist servers $S \neq S' \in \mathcal{S}$, such that $H(\mathsf{id}_S) = H(\mathsf{id}_{S'})$. As server identifiers are assumed to be unique and the hash values are sampled uniformly from $\{0,1\}^{2\lambda}$, a union bound over all pairs of servers shows that

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

Before we proceed, 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 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*.

Game $\mathbf{G}_5$: This game is as $\mathbf{G}_4$, but in the beginning of the game, it samples a random token $T^* \xleftarrow{\$} \mathcal{T}$. Then, it aborts if this is not the token that is used in the target registration. As $\mathcal{A}$ obtains no information about T^*, it follows that

$$\mathbf{Adv}_4 = |\mathcal{T}| \cdot \mathbf{Adv}_5.$$

Game $\mathbf{G}_6$: This game is as $\mathbf{G}_5$, but we add another change. Namely, the game now holds a map $L_\sigma[\cdot,\cdot]$ which is initially empty and used as follows. Whenever the oracle Challenge$(\pi_{T^*}^{i,0}, \mathsf{id}_S, M)$ for the guessed token T^* is called and a message m has to be signed during the execution of this oracle, the game first checks if $L_\sigma[\mathsf{id}_S, m]$ is already defined. If it is not defined yet, it samples random coins $coins \xleftarrow{\$} \{0,1\}$ for the signing algorithm and computes the signature σ as in the real protocol, but using these random coins instead of deriving them as $\hat{H}(seed_{T^*}, m)$. Then, it stores $L_\sigma[\mathsf{id}_S, m] := \sigma$. To proceed, it uses $L_\sigma[\mathsf{id}_S, m]$. Note that this change is only conceptual. First, note that in previous games, for token T^* and a fixed server id_S, signatures deterministically depend on messages m. Therefore, signatures can be stored by the game and be reused as it is done in $\mathbf{G}_6$. Further, due to the abort that we introduced in $\mathbf{G}_3$, the coins $coins$ that are used for signing are distributed

uniformly at random to $\mathcal{A}$ when a message is signed for the first time. Note that a message that is signed always contains hash of the server identifier, which is uniquely bound to id_S by the change we introduced in $\mathbf{G}_6$. In summary, we have

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

We want to bound $\mathbf{Adv}_6$ using a reduction from the $\mathbf{uf\text{-}hrk\text{-}idx}1$ security of $\mathsf{SIG}_{H'}$. This is possible, as all keys involved for token T^* are rerandomizations of $\mathsf{pk}_{T^*,0}$, signing keys are only needed for signing, we only need one signature per message, and each signed message is prefixed with the hash of the server identifier. Further, the winning condition and the aborts that we introduced imply that a successful adversary forges a signature for a fresh message. Formally, we give a reduction $\mathcal{B}$, which is as follows.

- $\mathcal{B}$ gets as input a public key pk^*. It gets access to a signing oracle SignO and a randomness oracle RandO.
- $\mathcal{B}$ samples T^* as in $\mathbf{G}_6$ at random. Then, it sets $\mathsf{pk}_{T^*,0} := \mathsf{pk}^*$. It also samples $ch_{T^*}, seed_{T^*}$ as in the scheme and defines the revocation key $\mathsf{rk}_{T^*} := (\mathsf{pk}_{T^*,0}, ch_{T^*})$. For the other tokens $T \in \mathcal{T} \setminus T^*$ it generates keys honestly as in the scheme. Then, it passes all revocation keys $\mathsf{rk}_T, T \in \mathcal{T}$ to $\mathcal{A}$.
- $\mathcal{B}$ simulates oracles Start, Challenge, Complete and random oracles. To simulate random oracle $\hat{H}$, it uses its oracle RandO. More precisely, to answer a query $\hat{H}(\mathsf{pk}_{T^*,0}, ch_{T^*}, \mathsf{id}_S)$, it returns $\mathsf{RandO}(H(\mathsf{id}_S))$. For other queries it simulates $\hat{H}$ honestly using lazy sampling. To simulate oracle $\mathsf{Challenge}(\pi_T^{i,j}, \cdot, \cdot)$ for $T \neq T^*$, it proceeds as in $\mathbf{G}_6$. To simulate oracle $\mathsf{Challenge}(\pi_{T^*}^{i,j}, \cdot, \cdot)$ it uses its oracle SignO. Concretely,
  - Whenever $\mathcal{B}$ is needs to compute a signature according to game $\mathbf{G}_6$, it uses oracle SignO. More precisely, whenever the entry $L_\sigma[\mathsf{id}_S, m]$ is not defined yet, it sets $idx := H(\mathsf{id}_S)$, queries $\mathsf{RandO}(idx)$, and queries $\mathsf{SignO}(\bar{m}, idx)$, where $m = (H(\mathsf{id}_S), \bar{m})$. Note that each message that has to be signed can be written in this way. Also, the usage of the map L_σ ensures that each message is only queried once.
- After termination of $\mathcal{A}$, reduction $\mathcal{B}$ first finds the forged authentication and the corresponding target registration. If the target registration does not involve token T^*, $\mathcal{B}$ aborts as in $\mathbf{G}_6$.
- 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 thus also in the target registration, see $\mathbf{G}_6$), and rs be the challenge that is used.
- $\mathcal{B}$ returns $\sigma^* := \sigma, \quad idx^* := H(\mathsf{id}_S)$, and $m^* := H(rs)$.

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

$$\mathbf{Adv}_6 \leq \mathbf{Adv}^{\mathcal{B}}_{\mathbf{uf\text{-}hrk\text{-}idx}1,\mathsf{SIG}_{H'}}.$$

□