

# Security Analysis of Signature Schemes with Key Blinding

Edward Eaton  
National Research Council Canada  
Ottawa, Ontario, Canada  
edward.eaton@nrc-cnrc.gc.ca

Tancrede Lepoint<sup>†</sup>  
Amazon Web Services  
New York, New York, USA  
tlepoint@amazon.com

Christopher A. Wood  
Cloudflare  
San Francisco, California, USA  
caw@heapingbits.net

## ABSTRACT

Digital signatures are fundamental components of public key cryptography. They allow a signer to generate verifiable and unforgeable proofs—signatures—over arbitrary messages with a private key, and allow recipients to verify the proofs against the corresponding and expected public key. These properties are used in practice for a variety of use cases, ranging from identity or data authenticity to non-repudiation. Unsurprisingly, signature schemes are widely used in security protocols deployed on the Internet today.

In recent years, some protocols have extended the basic syntax of signature schemes to support *key blinding*, a.k.a., key randomization. Roughly speaking, key blinding is the process by which a private signing key or public verification key is blinded (randomized) to hide information about the key pair. This is generally done for privacy reasons and has found applications in Tor and Privacy Pass.

Recently, Denis, Eaton, Lepoint, and Wood proposed a technical specification for signature schemes with key blinding in an IETF draft. In this work, we analyze the constructions in this emerging specification. We demonstrate that the constructions provided satisfy the desired security properties for signature schemes with key blinding. We experimentally evaluate the constructions and find that they introduce a very reasonable 2-3x performance overhead compared to the base signature scheme. Our results complement the ongoing standardization efforts for this primitive.

## 1 INTRODUCTION

Digital signature is a fundamental primitive used in the design of modern applications. Like their counterpart handwritten signatures, digital signatures provide a number of useful properties for applications, including *verifiability* and *unforgeability*. Informally, these properties mean that a signature computed over a message  $m$  with signing key  $sk$  can convince a verifier with overwhelming probability that it was signed by a participant with the corresponding public key  $pk$ . As such, digital signature schemes, or simply signature schemes, have a wide variety of applications, ranging from identity or data authenticity to non-repudiation. They are used in a wide assortment of Internet security protocols and technologies, including, though certainly not limited to, TLS and QUIC for secure network connections [10], DNSSEC for DNS record authenticity and integrity [4], and WebAuthn for authentication in web applications [1].

**Signature Schemes with Key Blinding.** A digital signature scheme allows someone in possession of a public key, message, and signature computed over that message to verify the validity of the signature. This functionality allows anyone in possession of the public key to *link* two (message, signature) pairs signed under the same private key together. Indeed, in most cases, this is a desired

property. Consider, for example, the use of signatures in WebAuthn. The server, which verifies authentication credentials presented by clients (containing a public key and signature), must necessarily learn when two credentials belong to the same user account so that the right application account is associated with the credential.

However, this link between two (message, signature) pairs may not always be desirable. In some settings, it is useful to produce signatures with a given key pair  $(sk, pk)$  such that the resulting signature is not linkable to  $pk$  without knowledge of a particular witness. That is, given  $pk$  corresponding to  $sk$ , witness  $r$ , and a message signature  $\sigma$ , one can determine if the signature was indeed produced using  $sk$ . In effect, the witness “blinds” the key pair associated with a message signature.

The desired goal of this blinding step is to ensure that a verifier cannot distinguish between two signatures produced from the same long term  $sk$  and two signatures produced from distinct keys  $sk$  and  $sk'$ . In other words, two independently blinded public keys and signatures are *unlinkable*. We refer to schemes with this type of functionality as signature schemes with key blinding.

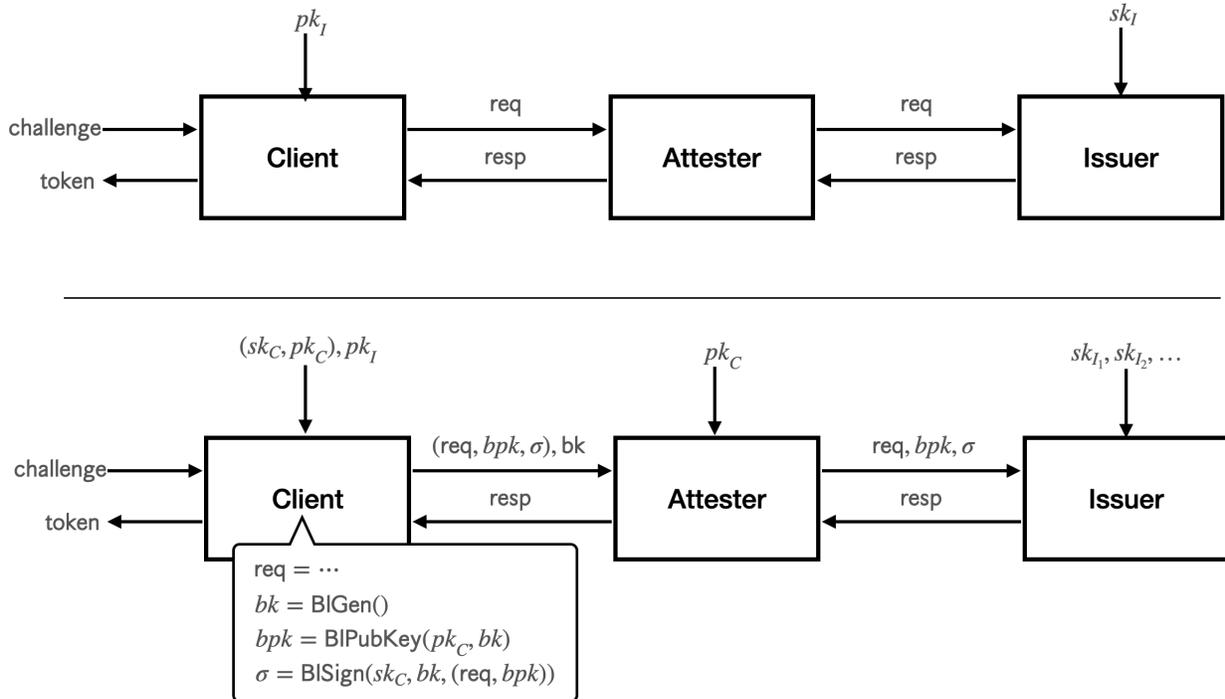
**Use Cases.** There are a variety of application use cases that motivate this type of primitive. Indeed, some applications have already deployed variants this primitive in practice, motivating the need for security analysis. We describe some of these application use cases below.

**Tor Hidden Services.** In Tor hidden services [23], each hidden service has a long-term identity key pair they used to sign data such as the service descriptors necessary to connect to the service. The identifier of a service is the public key used to verify this descriptor. To prevent long-term persistent blocking of any service based on its public key identifier, Tor derives per-epoch keys that are used for signing and verifying service descriptors. Roughly speaking, in epoch  $i$ , a service with long-term key pair  $(sk, pk)$  blinds its public key using a fresh blinding key  $bk$ . As a result, knowledge of  $pk$ , the blinding key  $bk$ , and the epoch lets one verify any signature produced for the epoch and associate it with a long-term service identity, whereas this is not possible without that knowledge [19].

**Private Airdrop.** Private airdrop for cryptocurrencies is another application of signature schemes with key blinding. Airdrop is a procedure for bootstrapping new cryptocurrency applications, wherein a sender gives away currency for recipients to let them join the system. Private airdrop hides the recipient of the airdrop in order to further entice users to join without compromising their privacy or safety [24].

At a high level, private airdrop works by having the sender generate a blinding key  $bk$ , blind the recipients public key  $pk$  under  $bk$  to produce  $bpk$ , a blinded public key, sign the message  $m$  to be placed on the cryptocurrency blockchain to produce  $\sigma$ , and then publish the  $(m, \sigma, bpk)$ . The sender also sends  $bk$  to the recipient out of band. Anyone can verify  $\sigma$  over  $m$  using  $bpk$ . However, only

<sup>†</sup> Work done while employed at Apple, Inc, prior to joining Amazon.



**Figure 1: Top: basic Privacy Pass issuance. Bottom: rate-limited Privacy Pass issuance using signatures with key blinding. The key pair  $(sk_I, pk_I)$  correspond to the blind signature protocol used in Privacy Pass, not the signature scheme with key blinding.**

the intended recipient that received  $bk$  out of band can validate that  $bpk$  is the blinded representation of  $pk$  with respect to blind  $bk$ .

**Rate-Limited Privacy Pass.** In Privacy Pass [6, 7], clients interact with an attester and issuer service to produce tokens, which are simply signatures over a client-chosen nonce. For privacy reasons, tokens are computed using a blind signature protocol [9] so that only the client learns the output signature and the attester and issuer learn nothing. The rate-limited version of Privacy Pass [18] aims to extend this basic version of Privacy Pass with the ability for the attester to limit the number of tokens clients request for specific websites, without the attester learning which website a specific client is interacting with. (Revealing that information to the attester would allow the attester to collect the client’s browsing history.) As such, the attester requires a unique identifier that is based on a per-client identifier and a per-website identifier for accounting purposes, e.g., so it can keep track of a monotonically increasing counter whenever a token is requested for that identifier.

To prevent the attester from masquerading as a client and requesting a token for a website of its choosing – effectively a *dictionary attack* to learn the websites a client visits – clients sign the token requests to the issuer with a secret key they know. Their identifier then becomes the corresponding public key. Importantly, to ensure that the issuer cannot use this public key and signature to link any two token requests to the same client, clients sign their requests with a freshly chosen blind. This lets the issuer check each request for validity without letting the attester forge requests on behalf of a

client; this interaction is shown in Figure 1. Apple’s Private Access Tokens system uses the split Attester and Issuer model of Privacy Pass [2]. In addition to the basic issuer, this system allows issuers to register for supporting the rate-limited variant [3].

**Standardization and Applicability.** Given the wide variety of existing and possible applications of this signature schemes with key blinding, the Crypto Forum Research Group (CFRG) recently started working on specifying them for practical applications. The draft specification [8] includes full details for two concrete signature schemes – one of which is based on Ed25519, a variant of EdDSA specified in [20], and another variant based on ECDSA [5]. While EdDSA and Schnorr signatures in general are often simpler, support for ECDSA is widespread enough in practice that including support for both was preferred by the community.

The straightforward way to extend a signature scheme over elliptic curves in particular with key blinding support is to sample a random key and then compute the blinded key as the scalar product of the signing key and blinding key. This technique lends itself naturally to EdDSA as described in [8] and as analyzed in this work, ECDSA is required new approaches. In particular, Morita et al. [21] demonstrate (in Section 4.2) that a related key attack on ECDSA (and DSA in general) which can arise if there exists such a linear relationship between the long-term signing key and blinding key. Extending ECDSA with key blinding support therefore required mitigating this related key attack. This paper analyzes the construction proposed in [8] and proves that it achieves the desired security properties.

As such, this paper complements the ongoing work to specify signature schemes with key blinding support for EdDSA and ECDSA. This is an important step to ensure safety and correctness of the techniques in the specification and will help enable wider applicability. Moreover, given that both EdDSA and ECDSA techniques are actively deployed in practice right now, analysis of these schemes has implications for real world systems.

**Our Approach and Contributions.** In this paper, we provide security analysis for a standardized mechanism being used in practice and in emerging privacy-enhancing technology standards. We formalize the syntax and security definitions for signature schemes with key blinding and unblinding, and formally prove security *unforgeability* and *unlinkability* for two concrete schemes based on EdDSA and ECDSA. The EdDSA construction matches that which is widely deployed in applications such as Tor and the ECDSA construction – which is new – matches that which is deployed in applications such as rate-limited Privacy Pass. We also experimentally evaluate the computational cost these extensions compared to the base signature schemes. Our results indicate that they contribute modest overhead compared to the base schemes, dominated by the cost of additional private key operations, yielding 2x to 3x overhead for EdDSA and ECDSA, respectively.

**Outline.** The rest of this paper is organized as follows. Section 2 describes the formal syntax and security definitions for signature schemes with key blinding. Section 3 presents a description of the EdDSA and ECDSA variants. Sections 4 and 5 present the formal security analysis of EdDSA and ECDSA variants, respectively. Section 6 presents our experimental benchmarks. Finally, Section 7 describes related work and Section 8 concludes.

## 2 SYNTAX AND SECURITY DEFINITIONS

In this section we describe the syntax and security definitions for signature schemes with key blinding as described in 1.

**Syntax.** A digital signature scheme is a tuple of three algorithms:

$$(\text{Sig.KeyGen}, \text{Sig.Sign}, \text{Sig.Verify})$$

$\text{Sig.KeyGen}$  generates a private and public key pair,  $(sk, pk)$ . One can then invoke  $\text{Sig.Sign}$  given signing key  $sk$  and a message  $m$  to produce a signature  $\sigma$ .  $\text{Sig.Verify}$  accepts as input public key  $pk$ , message  $m$ , and signature  $\sigma$  and outputs 1 if the signature is valid, and 0 otherwise. A signature scheme is correct if for all  $m \in \{0, 1\}^*$ , it holds that

$$\Pr \left[ \text{Sig.Verify}(pk, m, \sigma) = 1 \mid \begin{array}{l} (sk, pk) \leftarrow \text{Sig.KeyGen}() \\ \sigma \leftarrow \text{Sig.Sign}(sk, m) \end{array} \right] = 1 ..$$

A signature schemes with key blinding extends the syntax of a digital signature scheme the following algorithms:

$$(\text{BK.BIGen}, \text{BK.BIPubKey}, \text{BK.BISign}) .$$

$\text{BK.BIGen}$  generates a blinding key  $bk$ .  $\text{BK.BIPubKey}$  takes as input a public key  $pk$ , a blinding key  $bk$ , and a context  $ctx$ , and outputs a blinded public key  $bpk$ .  $\text{BK.BISign}$  takes as input a private key  $sk$ , blinding key  $bk$ , a context  $ctx$ , and a message  $m$ , and produces a signature  $\sigma$ . Correctness requires that for all messages  $m \in \{0, 1\}^*$  and contexts  $ctx \in \{0, 1\}^*$ , it holds that

$$\Pr \left[ \text{BK.Verify}(bpk, m, \sigma) = 1 \mid \begin{array}{l} (sk, pk) \leftarrow \text{BK.KeyGen}() \\ bk \leftarrow \text{BK.BIGen}() \\ bpk \leftarrow \text{BK.BIPubKey}(pk, bk, ctx) \\ \sigma \leftarrow \text{BK.BISign}(sk, bk, m, ctx) \end{array} \right] = 1.$$

Optionally, a signature scheme with key blinding may allow one to *unblind* a public key with respect to its blinding key, yielding the original (unblinded) public key. This function is denoted by  $\text{BK.UnblindPublicKey}$ . Correctness requires that for every context  $ctx \in \{0, 1\}^*$ , any key pair  $(sk, pk)$  and blinding key  $bk$  output by  $\text{BIGen}$ , it holds that

$$\Pr[\text{UnblindPublicKey}(\text{BIPubKey}(pk, bk, ctx), bk, ctx) = pk] = 1.$$

**Unlinkability.** A signature scheme with key blinding is said to be unlinkable if, informally, an adversary without knowledge of the long-term public key who observes many blinded public key, and signatures that verify under those blinded public keys, cannot distinguish between a blinding of the long-term public key, or a blinding of a freshly-generated public key. This is formally captured in Fig. 2.

We define the advantage  $\text{Adv}_{\mathcal{A}}^{\text{unlinkable}}$  an adversary has in winning this game as:

$$\left| \Pr[\text{UNLINKABILITY}^{\mathcal{A}}(\lambda) = 1] - \Pr[\text{UNLINKABILITY}^{\mathcal{A}}(\lambda) = 0] \right|.$$

Note that this formulation of unlinkability differs slightly from ones previously seen in the literature [11]. In the other version of unlinkability, the adversary is allowed to query the  $\text{BLPUBLICKEY}$  oracle with a  $bk$  of their choice, receiving the corresponding blinded public key, instead of a public key blinded with a random and unknown  $bk$ . The reason for the difference is that we are interested in establishing the unlinkability of the scheme when the key-blinding scheme admits an unblinding functionality. If the scheme allows for unblinding, then the blinding key  $bk$  must be treated as privileged information and not allow the adversary to see or control it. The differences between one-way and bi-directional blinding are discussed further in Section A.1.

In order to establish unlinkability for Ed25519, we follow the proof technique of Eaton et al. [11]. In particular, we show that two conditions hold: (1) an adversary with access to a blinding oracle and signing oracle cannot distinguish between a new blinding of a long-term key and a blinding of a freshly-chosen key (the ‘independent blinding’ property), and (2) that signatures with an identical distribution that are produced from blinded public keys depends only on the blinded public key and not on long-term public key. Intuitively, the former property means that the blinded public key is *independent* from the long-term public key, whereas the latter means that signatures leak no information about the long-term signing key. Eaton et al. previously established that these two properties are sufficient for unlinkability when one-way blinding is employed. In Appendix B we show that these properties are also sufficient for bidirectional blinding. Then, in Section 4.1 we establish these two properties for Ed25519.

For ECDSA, we take a more direct approach to establishing unlinkability guided by the way that blinding in ECDSA entirely re-randomizes both the public key and secret key space (Section 5.1).

<p><b>Game UNLINKABILITY<math>^{\mathcal{A}}(\lambda)</math></b></p> <hr/> 1: $\Sigma \leftarrow \emptyset$ 2: $sk_S, pk_S \leftarrow \text{KeyGen}()$ 3: $s = 0$ 4: $b \leftarrow \{0, 1\}$ 5: $b' \leftarrow \mathcal{A}^{\text{BLPUBLICKEY, CHALLENGE, BLKEYSIGN}}()$ 6: <b>return</b> $b = b'$
<p><b>Oracle BLPUBLICKEY()</b></p> <hr/> 1: $bk \leftarrow \text{BlGen}()$ 2: $bpk = \text{BlPubKey}(pk_S, bk)$ 3: $\Sigma \leftarrow \Sigma \cup \{(bpk, bk)\}$ 4: <b>return</b> $bpk$
<p><b>Oracle CHALLENGE()</b></p> <hr/> 1: <b>if</b> $s = 1$ ; <b>return</b> $\perp$ 2: $s = 1$ 3: <b>if</b> $b = 0$ 4: $sk'_S, pk'_S \leftarrow \text{KeyGen}()$ 5: $sk_S^* \leftarrow sk'_S, pk_S^* \leftarrow pk'_S$ 6: <b>else</b> 7: $sk_S^* \leftarrow sk_S, pk_S^* \leftarrow pk_S$ 8: $bk^* \leftarrow \text{BlGen}()$ 9: $bpk^* = \text{BlPubKey}(pk_S^*, bk^*)$ 10: $\Sigma \leftarrow \Sigma \cup \{(bpk^*, bk^*)\}$ 11: <b>return</b> $bpk^*$
<p><b>Oracle BLKEYSIGN(<math>m, bpk</math>)</b></p> <hr/> 1: <b>if</b> $s = 1 \wedge bpk = bpk^*$ ; <b>return</b> $\text{BlSign}(sk_S^*, bk^*, m)$ 2: <b>if</b> $(bpk, bk) \notin \Sigma$ ; <b>return</b> $\perp$ 3: <b>return</b> $\text{BlSign}(sk_S, bk, m)$

**Figure 2: Unlinkability security game for a signature scheme with key blinding.**  $\mathcal{A}$  is given access to all oracles in the game.

This approach would likely also work for Ed25519, but we have elected to use the framework for Ed25519 as it is more generic and teases apart the properties of signatures being independent of the long-term secret key.

**Unforgeability.** Unforgeability is mostly the same as it is for classical digital signatures, with one major change: signing requests and forgeries are made with respect to blinded public keys chosen by the adversary.

Strong unforgeability is defined in Figure 3.<sup>1</sup> It mostly resembles the standard game of strong unforgeability, with a few key differences introduced by blinding. The adversary is given the public key  $pk$ . They are able to make adaptive signing queries with respect to any blind key  $bk$  of their choosing, including no blind key at all (indicated with  $bk = \perp$ ). Eventually, they must submit a forgery,

<sup>1</sup>Note that the existential variant of this game in the context of ECDSA was defined in [17, Sec. 6].

<p><b>Game UNFORGEABILITY<math>^{\mathcal{A}}(\lambda)</math></b></p> <hr/> 1: $\Sigma \leftarrow \emptyset$ 2: $sk, pk \leftarrow \text{KeyGen}()$ 3: $(m^*, bk^*, \sigma^*) \leftarrow \mathcal{A}^{\text{BLKEYSIGN}}(pk)$ 4: <b>if</b> $bk^* = \perp$ 5: $pk^* \leftarrow pk$ 6: <b>else</b> 7: $pk^* \leftarrow \text{BlPubKey}(pk, bk^*)$ 8: <b>if</b> $\text{Verify}(pk^*, m^*, \sigma^*)$ 9: <b>if</b> $(bk^*, m^*, \sigma^*) \notin \Sigma$ 10: <b>return</b> <i>True</i> 11: <b>return</b> <i>False</i>
<p><b>Oracle BLKEYSIGN(<math>m, bk</math>)</b></p> <hr/> 1: <b>if</b> $bk = \perp$ 2: $\sigma \leftarrow \text{Sign}(sk, m)$ 3: <b>else</b> 4: $\sigma \leftarrow \text{BlSign}(sk, bk, m)$ 5: $\Sigma \leftarrow \Sigma \cup \{(bk, m, \sigma)\}$ 6: <b>return</b> $\sigma$

**Figure 3: Signature scheme strong unforgeability security game.**

consisting of a message, a signature, and an optional blind key. If the signature verifies with respect to the (blinded) public key, and that exact signature was not returned from the BLKEYSIGN with the same  $m^*, bk^*$  input, then the adversary wins the game.

This definition considers any tuple  $(m^*, bk^*, \sigma^*)$  for which  $\sigma^*$  was not the result of query to  $\text{BlSign}(m^*, bk^*)$  a valid forgery. In other words, if an adversary submits a signing query, and then is able to modify either  $\sigma^*$  or  $bk^*$  (or both) and still have an accepting signature, then they have won. This means that signatures are not malleable, and are uniquely tied to the  $bk$  for which they were issued.

One can also define weaker notions of unforgeability for key-blinding schemes. In one previous work [11], the authors only considered the signature scheme broken if the tuple  $(m^*, bk^*)$  was new—in other words, if an adversary modified the signature but not  $bk$  this did not count as a break. In another previous paper [14], the model only accepts forgeries if the message is entirely new. This leaves open the possibility that signatures can be modified or valid to verify under other  $bk$  values, which may or may not present an issue for protocols.<sup>2</sup>

In this work, we establish the strong unforgeability of Ed25519 as per the security notion in Figure 3. Additionally, we establish the existential unforgeability of ECDSA with key-blinding, i.e., unforgeability in the simpler model where only the tuple  $(bk^*, m^*)$  is verified to be in  $\Sigma$ . Indeed, plain ECDSA is known to not be strongly unforgeable: for any valid signature  $\sigma = (r, s)$ ,  $(r, -s)$  is

<sup>2</sup>An issue based on such malleability property, on the original variant of ECDSA with key blinding without the H2S hash function in Figure 5, was reported by Lepoint at <https://github.com/TFPau/privacy-proxy/issues/166>.

also a valid signature. Instead, plain ECDSA is proved to be strongly unforgeable *up to sign* in [17, Sec. 4.1.1]. Similarly, the analysis in Section 5 enables to deduce that ECDSA with key blinding is strongly unforgeable up to sign. We leave as an open problem whether ECDSA can be modified to meet the stronger security definition in Figure 3. In Section A.3 we discuss some of the design decisions that might enable this.

### 3 CONCRETE SIGNATURE SCHEMES

This section presents the constructions of signature with key blinding based on EdDSA and ECDSA proposed in [8]. The correctness of these constructions is immediate; we prove their unforgeability and unlinkability in Sections 4 and 5. Considerations that led to these concrete constructions is discussed in Section A.

#### 3.1 EdDSA with Key Blinding

The Ed25519 signature scheme standardized in RFC8032 [20] is a variant of the EdDSA Schnorr signature scheme. The vanilla variant of Ed25519 consists of three algorithms: KeyGen, Sign, and Verify. The input to KeyGen, referred to as a private key or seed, is a random 32 bytes.  $L$  is the order of the Ed25519 group, i.e.,

$$L = 2^{252} + 2774231777372353535851937790883648493,$$

and  $G$  is the generator for the group. ScalarClamp is a function that interprets its input as a little-endian integer and performs clamping. A description of Ed25519 with key blinding is shown in Fig. 4.

<b>KeyGen()</b>	<b>InnerKeyGen(<math>sk</math>)</b>
1: $sk \leftarrow \{0, 1\}^{256}$	1: $h \leftarrow \text{SHA512}(sk)$
2: $\perp, pk, \perp \leftarrow \text{InnerKeyGen}(sk)$	2: $x \leftarrow \text{ScalarClamp}(h[0 : 32])$
3: <b>return</b> $sk, pk$	3: $A \leftarrow xG$
	4: $pre \leftarrow h[32 : 64]$
	5: <b>return</b> $x, A, pre$
<b>BlSign(<math>sk, bk, m</math>)</b>	<b>BIPubKey(<math>pk, bk</math>)</b>
1: $x, A, pre \leftarrow \text{InnerKeyGen}(sk)$	1: $h \leftarrow \text{SHA512}(bk)$
2: $\beta, pre' \leftarrow \text{SHA512}(bk)$	2: $\beta \leftarrow h[0 : 32]$
3: $r \leftarrow \text{SHA512}((pre    pre'), m)$	3: $bpk \leftarrow \beta \cdot pk$
4: $R \leftarrow rG$	4: <b>return</b> $bpk$
5: $A' \leftarrow \beta \cdot A$	
6: $k \leftarrow \text{SHA512}(R, A', m)$	
7: $s \leftarrow (r + kx\beta) \bmod L$	
8: <b>return</b> $(R, s)$	
	<b>Verify(<math>pk, m, (R, s)</math>)</b>
	1: $k \leftarrow \text{SHA512}(R, A, m)$
	2: <b>return</b> $sG = R + kA$

Figure 4: Ed25519 with key blinding.

#### 3.2 ECDSA with Key Blinding

We describe the ECDSA with Key Blinding scheme over an elliptic curve  $E$  of prime order  $q$ , with generator  $G$ , using the notation of [17] in Fig. 5, denoted ECDSA. In particular, we denote  $\tilde{C}$  the

conversion function from an elliptic curve point to an element of  $\mathbb{Z}_q$ . The secret key for ECDSA is a random  $sk = d \in \mathbb{Z}_q^*$ , and the public key is  $pk = d \cdot G \in E$ . The key blinding BIPubKey operation with parameter  $bk$  can be applied on a public key  $pk$  as follows. If  $bk = \perp$ , set  $\beta = 1$ , otherwise set  $\beta = \text{H2S}(bk)$  where H2S is the `hash_to_field` from [12], and output  $\text{BIPubKey}(pk, bk) = \beta \cdot pk \in E$ .

<b>KeyGen()</b>	<b>BIGen</b>
1: $sk \leftarrow \mathbb{Z}_q^*$	1: <b>return</b> $bk \leftarrow \{0, 1\}^{256}$
2: $pk \leftarrow sk \cdot G$	
3: <b>return</b> $(sk, pk)$	
<b>InnerBlind(<math>bk</math>)</b>	<b>BIPubKey(<math>pk, bk</math>)</b>
1: <b>if</b> $bk = \perp$ <b>then</b> $\beta \leftarrow 1$	1: $\beta \leftarrow \text{InnerBlind}(bk)$
2: <b>else</b> $\beta \leftarrow \text{H2S}(bk) \in \mathbb{Z}_q^*$	2: <b>return</b> $\beta \cdot pk$
3: <b>return</b> $\beta$	
<b>BlSign(<math>sk, bk, m</math>)</b>	
1: $h \leftarrow \text{Hash}(m) \in \mathbb{Z}_q$	
2: $\beta \leftarrow \text{InnerBlind}(bk)$	
3: $r \leftarrow \mathcal{R} \in \mathbb{Z}_q^*$ ; $\mathcal{R} \leftarrow r \cdot G$	
4: $t \leftarrow \tilde{C}(\mathcal{R}) \in \mathbb{Z}_q$	
5: <b>if</b> $t = 0 \parallel h + t\beta sk = 0$ <b>then return fail</b>	
6: $s \leftarrow r^{-1}(h + t\beta sk)$	
7: <b>return</b> $(s, t)$	
<b>Verify(<math>bpk, m, (s, t)</math>)</b>	
1: $h \leftarrow \text{Hash}(m) \in \mathbb{Z}_q$	
2: $\mathcal{R} \leftarrow s^{-1}h \cdot G + s^{-1}t \cdot bpk$	
3: <b>if</b> $R \neq O \wedge \tilde{C}(\mathcal{R}) = t$ <b>then return true</b>	
4: <b>else return false</b>	

Figure 5: ECDSA with key blinding. The scheme makes use of two hash functions  $\text{Hash}: \{0, 1\}^* \rightarrow \mathbb{Z}_q$  and  $\text{H2S}: \{0, 1\}^* \rightarrow \mathbb{Z}_q^*$ .

## 4 SECURITY ANALYSIS OF EDDSA WITH KEY BLINDING

In this section, we analyze the security of the rate-limited issuance protocol components build on EdDSA, and in particular Ed25519. We focus on proving two properties: (1) unlinkability and unforgeability of EdDSA with key blinding, and (2) unforgeability and pseudorandomness of BK-PRF built on EdDSA.

### 4.1 Signature Unlinkability

We now proceed to show that EdDSA with key blinding as described in Section 3.1 is unlinkable. As discussed in Section 2 and expanded upon in Appendix B, showing that the key-blinding scheme is unlinkable can be reduced to establishing two properties: the *independent blinding* property, which states that blinding suitably re-randomizes the public key space, and *signing with oracle reprogramming*, which states that a public key and the ability to program

a random oracle is enough to simulate a signing oracle—in other words, that the distribution of signatures is dependent only on the public key.

First, we show the following.

**THEOREM 4.1 (EDDSA INDEPENDENT BLINDING).** *For any probabilistic polynomial time adversary  $\mathcal{A}$ , we have that  $\text{Adv}_{\mathcal{A}, \text{Ed25519}}^{\text{IndBlind}, n} \leq n \text{Adv}_{\mathcal{A}}^{\text{DDH}}$ , where  $\text{Adv}_{\mathcal{A}}^{\text{DDH}}$  is the adversary's advantage in winning the DDH game.*

To prove this, assume there exists an adversary  $\mathcal{A}$  that succeeds with advantage  $\text{Adv}_{\mathcal{A}, \text{Ed25519}}^{\text{IndBlind}, n} \geq \epsilon(\lambda)$  with a slightly modified version of BIPubKey, denoted ModBIPubKey. In particular, rather than BIPubKey outputting a single public key  $pk_R$  given input key  $pk$  and blind key  $bk$ , ModBIPubKey invokes BIPubKey on input key  $pk$  and blind key  $bk$  and returns  $pk_R$ , but it also returns  $\text{pub}(bk)$  – the public blind key corresponding to the scalar used to blind  $pk$ . For Ed25519 as described in 4, this is  $\beta G$ , where  $\beta \parallel \text{pre}' = \text{SHA512}(bk)$ . Given  $y = \text{ModBIPubKey}(pk, bk)$ , we denote  $y[0]$  as  $\text{pub}(bk)$  and  $y[1]$  as  $pk_R$ .

Note that this is strictly more information than the adversary would learn when interacting with BIPubKey directly. Now, let  $(\perp, pk_0) = \text{Ed25519.KeyGen}()$ , and  $(\perp, pk_i) = \text{Ed25519.KeyGen}()$  and  $(bk_i, \perp) = \text{Ed25519.KeyGen}()$  for  $i = 1, \dots, n$ . Consider the following distribution output from ModBIPubKey invoked upon a fixed public key  $(pk_0)$ :

$$L_1, \dots, L_n = \text{ModBIPubKey}(pk_0, bk_1), \dots, \text{ModBIPubKey}(pk_0, bk_n)$$

And the following distribution output from ModBIPubKey invoked upon freshly generated public keys  $(pk_i)$ :

$$R_1, \dots, R_n = \text{ModBIPubKey}(pk_1, bk_1), \dots, \text{ModBIPubKey}(pk_n, bk_n)$$

Importantly, both distributions use the same  $bk_i$ . For  $i = 0, \dots, n$ , let  $X^{(i)} = (L_1, \dots, L_{n-i}, R_{n-i+1}, \dots, R_n)$  be the distribution defined in terms of  $L$  and  $R$ . Observe that  $X^{(0)} = R_1, \dots, R_n$  and  $X^{(n)} = (L_1, \dots, L_n)$ . Assume, towards a contradiction, that  $\Pr[\mathcal{A}(X^{(0)}) = 1] - \Pr[\mathcal{A}(X^{(n)}) = 1] \geq \epsilon(\lambda)$ . By the triangle inequality, it follows that there exists an  $i^*$  for which  $\Pr[\mathcal{A}(X^{(i^*-1)}) = 1] - \Pr[\mathcal{A}(X^{(i^*)}) = 1] \geq \frac{\epsilon(\lambda)}{n}$ . The difference between  $X^{(i^*-1)}$  and  $X^{(i^*)}$  effectively simplifies to a difference between

$$L_{i^*} = \text{ModBIPubKey}(pk_0, bk_{i^*})$$

and

$$R_{i^*} = \text{ModBIPubKey}(pk_{i^*}, bk_{i^*})$$

Given a distinguisher  $\mathcal{D}$  between  $L_{i^*}$  and  $R_{i^*}$ , one can construct a distinguisher  $\mathcal{D}'$  between the DDH triples  $(pk_0, L_{i^*}[0], L_{i^*}[1])$  and  $(pk_0, R_{i^*}[0], R_{i^*}[1])$ . Thus, we can bound this distinguishing advantage  $\text{Adv}_{\mathcal{A}}^{\text{DDH}}$ , and therefore  $\text{Adv}_{\mathcal{A}, \text{Ed25519}}^{\text{IndBlind}, n} \leq n \text{Adv}_{\mathcal{A}}^{\text{DDH}}$ .

This establishes unlinkability against an adversary who does not make any signing queries and  $n$  queries to the blinding oracle. To account for the BLKEYSIGN oracle, use Lemma 1 from [11], which requires that we show that Ed25519 permits signing with oracle preprogramming. To prove this, upon signing message  $m$ , we construct our simulator as follows. First, it randomly samples two scalars  $(k, s)$ , computes  $R = sG - kA'$ , and then reprograms the

random oracle (SHA512 for Ed25519) such that  $\text{SHA512}(R, A', m) = k$ , where  $A'$  the blinded public key used to verify the message signature.

Since the adversary controls  $m$  but not the signing keys  $sk$  or  $bk$ , the min-entropy this simulator's output is dependent only on  $bk$ , or  $2^{-q}$ , where  $q$  is the order of the Ed25519 group. (The size of prefix exceeds  $\log_2(2^q)$ .) Moreover, since the random oracle was preprogrammed such that the simulator's output is valid, the resulting distribution of output signatures is the same as the real signing protocol. Therefore the statistical difference from Definition B.2  $\delta_{\text{Ed25519}} = 0$ .

Lastly, note that the adversary cannot determine when oracle preprogramming occurs since that would have required them to query  $\text{SHA512}(R, A', m)$  before signing occurred, which cannot happen as  $R$  is computed based on a random  $r$ . The minimum entropy of  $R$  is less than  $\log_2(1/L + 1/2^{259})$ , where the  $1/2^{259}$  term comes from the slight 'wraparound' of  $r$  when taken modulo  $L$ , since  $r$  is a uniform 512 bits.

Thus, from Lemma 1 in [11], we have the following unlinkability result.

**THEOREM 4.2 (EDDSA UNLINKABILITY).** *For any probabilistic polynomial time adversary  $\mathcal{A}$  that makes  $q_S$  signing queries,  $n$  blinding queries, and  $q_H$  random oracle queries, it holds that*

$$\text{Adv}_{\mathcal{A}, \text{Ed25519}}^{\text{unlinkability}} \leq q_H q_S \left( \frac{1}{L} + \frac{1}{2^{259}} \right) + n \text{Adv}_{\mathcal{A}}^{\text{DDH}}.$$

## 4.2 Signature Unforgeability

Strong unforgeability is defined in Figure 3. It mostly resembles the standard game of strong unforgeability, with a few key differences introduced by masking. The adversary is given the public key  $A$ . They are able to make adaptive signing queries with respect to any mask of their choosing, including no mask at all. Eventually, they must submit a forgery, consisting of a message, a signature, and an optional mask. If the signature verifies with respect to the (masked) public key, and that exact signature was not returned from the BLKEYSIGN oracle with the same  $m, bk^*$  input, then the adversary wins the game.

To establish the strong unforgeability of Ed25519 with key-blinding, we show that an adversary who can win the game in Figure 3 implies the existence of an adversary who can break the security of 'raw', or unblinded Ed25519 with SHA512 replaced with a random oracle. The unforgeability of Ed25519 with key-blinding has been considered before, in a draft posted by Hopper to the Tor mailing list [19]. Our proof builds on Hopper's in a few key ways. First, our formulation of unforgeability is generalized and separated from the Tor context. It allows the adversary to make signing queries and forgeries with respect to any blinding key, or optionally no blinding key at all. Our proof is also tighter, and takes into close consideration some of the details of Ed25519 (such as secret key clamping and the fact that the output of the hash function is not uniform over the order of the group).

To show the security reduction, we need to establish an algorithm  $\mathcal{B}$  that has access to a random oracle  $H_{\mathcal{B}}$ , a public key  $A$ , and an adversary  $\mathcal{A}$  that can win the game in Figure 3. Our algorithm  $\mathcal{B}$  will simulate a random oracle  $H_{\mathcal{A}}$  for the adversary  $\mathcal{A}$ . The central idea is to program  $H_{\mathcal{A}}$  so that a relationship between it and  $H_{\mathcal{B}}$  is

strictly enforced. This relationship allows the security reductions to translate between valid signatures made with respect to  $H_{\mathcal{B}}$  and a blinded public key  $A'$  and a signature that is valid with respect to  $H_{\mathcal{A}}$  and an unblinded public key  $A$ .

To see how this can be done, consider the forgery  $(m^*, bk^*, (R^*, s^*))$ . If we let  $\beta$  be the scalar so that  $\beta A = \text{Ed25519.BIPubKey}(A, bk^*)$ , then the verification process insists that  $s^*G = R^* + H_{\mathcal{A}}(R^*, \beta A, m^*)\beta A$ . We will program the random oracle  $H_{\mathcal{A}}$  so that  $H_{\mathcal{A}}(R^*, \beta A, m^*) \equiv \beta^{-1}H_{\mathcal{B}}(R^* + v\beta A, A, m^*) + v \pmod{L}$  for a random scalar  $v$ . We can then see that

$$\begin{aligned} s^*G &= R^* + H_{\mathcal{A}}(R^*, \beta A, m^*)\beta A \\ &= R^* + \left(\beta^{-1}H_{\mathcal{B}}(R^* + v\beta A, A, m^*) + v\right)\beta A \\ &= R^* + v\beta A + H_{\mathcal{B}}(R^* + v\beta A, A, m^*)A, \end{aligned} \quad (1)$$

making  $(R^* + v\beta A, s^*)$  a valid signature of  $m^*$  under the public key  $A$  with respect to the random oracle  $H_{\mathcal{B}}$ .

**THEOREM 4.3 (ED25519 UNFORGEABILITY).** *Let  $\mathcal{A}$  be an adversary capable of winning the Ed25519 unforgeability security game in Figure 3 with  $q_H$  queries to a random oracle  $H_{\mathcal{A}}$  and  $q_S$  queries to the oracle  $\text{BLKEYSIGN}$  with probability  $p$ . Then there exists an adversary  $\mathcal{B}$  who can break the strong unforgeability of Ed25519 without key-blinding in  $q_H$  queries to a random oracle  $H_{\mathcal{B}}$  and  $q_S$  queries to a signing oracle, with probability at least*

$$p - \frac{8q_H^2 + 16q_H + 11}{2^{259}} - \frac{q_H^2 + 2q_Hq_S + 1}{L}. \quad (2)$$

**PROOF.** We will use a game-hopping proof to establish Theorem 4.3. For assistance in following the proof, we present two figures showing pseudocode for the evolution of how queries are handled. Figure 6 presents how the oracle operates in the final game,  $G_7$ . Line comments indicate in which game each line of code is added. Note that we define three routines for the random oracle depending on the nature of the input. Originally all three are the same (responding with a uniform 512 bits), but change as game hops are introduced. Note as well that we only consider new queries. In all cases, if a query is repeated, an identical response is provided. In Figure 7 we present similar code for the signing oracle. As the way the signing oracle operates changes much more, we present the pseudocode in its entirety for games where it changes—games  $G_6$  and  $G_7$ .

Game  $G_0$  corresponds to the unforgeability game in Figure 3. This means that the adversary has access to a  $\text{BLKEYSIGN}$  oracle that takes in a message and an (optional) blinding key, and computes a signature by way of  $\text{BISign}$ . The adversary also has access to a random oracle  $H_{\mathcal{A}}$ , which replaces all instances of  $\text{SHA512}$ . Because the adversary is explicitly given the identity public key in the unforgeability game, there is no reason to provide them with a  $\text{BLPUBLICKEY}$  oracle—they may compute the functionality themselves for any blinding key of their choice. However, computing  $\text{BIPubKey}$  still requires a query to the random oracle.

In game  $G_1$  we will abort if the adversary ever queries a  $bk$  to  $H_{\mathcal{A}}$  such that the resulting blinding,  $A'$  is equal to the identity point 0. Recall that we do not use the clamping procedure for the blinding key. So the only way for the adversary to get a blinding to the identity point is if the  $\beta$  value that is generated is a multiple of

$L$ . This happens with probability close to  $\frac{1}{L}$ , but since we have a uniform 256 bits and  $L$  does not divide  $2^{256}$ , we also add a  $1/2^{256}$  term to denote the probability of hitting the ‘last’ multiple of  $L$  (this does not compute the exact probability, but provides a simple upper bound). Therefore,  $|\Pr_{G_0}[\mathcal{A} \text{ wins}] - \Pr_{G_1}[\mathcal{A} \text{ wins}]| \leq \frac{1}{L} + \frac{1}{2^{256}}$ .

In game  $G_2$ , we will modify the game to abort if the adversary ever “predicts” a blinded public key without first constructing that key by making the requisite query to  $H_{\mathcal{A}}$ . Specifically, in the signing (and verification) routine, the tuple  $(R, A', m)$  is queried, where  $R$  is the commitment point,  $A'$  is the (blinded) public key, and  $m$  is the message. In game  $G_2$ , we will maintain a list of blindings that the adversary has observed, and abort if they first query a tuple  $(R, A', m)$  for which  $A'$  has not been blinded to, and later query a  $bk$  to  $H_{\mathcal{A}}$  that results in a blinding to  $A'$ .

To do this, we maintain a table  $T_1$  of the blindings that the adversary has knowledge of. When the adversary makes a query  $H_{\mathcal{A}}(q)$  and  $q$  can be parsed as a blinding key (i.e., it is  $\lambda$  bits), then (assuming this is the first time the query has been made) we sample a response  $h \leftarrow \{0, 1\}^{512}$  and calculate  $\beta \leftarrow h[0 : 32]$  and  $A' \leftarrow \beta \cdot A$ . Record  $(q, h, A')$  in a table  $T_1$ , and return  $h$  to  $\mathcal{A}$ .

When the adversary makes a query of the form  $(R, A', m)$  (that is, where  $R$  and  $A'$  are points on the curve and  $m$  is an arbitrary-length message), we check and see if  $A'$  appears in the third column of table  $T_1$ . If it does not appear in the table, then we add it to a list  $U$  of unexplained blinded public keys. If, when adding entries into table  $T_1$ , we ever add a public key  $A'$  that appears on the list of unexplained blinded public keys, we abort the game.

To consider the probability that we abort, we need to consider the chance that a query the adversary makes to  $H_{\mathcal{A}}$  causes an entry to  $T_1$  that is on the list  $U$ . We can divide the adversary’s  $q_H$  queries to  $H_{\mathcal{A}}$  queries into the  $q_{H,1}$  that are just  $\lambda$  bits and the  $q_{H,2}$  of the form  $(R, A', m)$ . So we can see that if all  $q_{H,2}$  queries had different  $A'$  values and all added an a different  $A'$  to  $U$ , then when a query is made to add an entry to  $T_1$ , the chances that it will ‘hit’ one of the  $q_{H,2}$  values is (similar to the chances of hitting the identity point),  $\frac{q_{H,2}}{L} + \frac{q_{H,2}}{2^{256}}$ . So with  $q_{H,1}$  such queries, the overall probability is bounded by  $q_{H,1}q_{H,2} \left(\frac{1}{L} + \frac{1}{2^{256}}\right)$ . Since  $q_{H,1}$  and  $q_{H,2}$  must both be less than or equal to  $q_H$ , this probability is bounded by  $q_H^2 \left(\frac{1}{L} + \frac{1}{2^{256}}\right)$ . Thus  $|\Pr_{G_2}[\mathcal{A} \text{ wins}] - \Pr_{G_1}[\mathcal{A} \text{ wins}]| \leq \frac{q_H^2}{L} + \frac{q_H^2}{2^{256}}$ .

In Game  $G_3$  we change how responses to the random oracle are generated when the adversary makes a query of the form  $(R, A', m)$ . At this point, we introduce a random oracle for the reduction,  $H_{\mathcal{B}}$ . In the final game, when we show how to construct a forgery for plain Ed25519 signatures, it will be with respect to the random oracle  $H_{\mathcal{B}}$ . On input of a query  $q = (R, A', m)$ , instead of responding with a uniformly random  $h \leftarrow \{0, 1\}^{512}$ , we do the following:

- (1) Perform a lookup on table  $T_1$  to see if  $A'$  appears in the last column of an entry  $(q, h, A')$ . If so, then take  $\beta \leftarrow h[0 : 32]$  and sample  $v \leftarrow \{0, \dots, L-1\}$ . If there is no entry in  $T_1$  with  $A'$  in the last column, simply respond with a random response  $y \leftarrow \{0, 1\}^{512}$ , and add  $(q, y, \perp)$  to  $T_2$ . If the adversary is making a query with the unblinded public key (so  $A' = A$ , set  $\beta \leftarrow 1$ ).
- (2) Compute  $R + vA'$  and query  $h \leftarrow H_{\mathcal{B}}(R + vA', A, m)$ .
- (3) Compute  $z \leftarrow \beta^{-1}h' + v \pmod{L}$ .

$H_{\mathcal{A}}(q = bk)$		$H_{\mathcal{A}}(q = (R, A', m))$	
1: $h \leftarrow_{\$} \{0, 1\}^{512}$	// All	1: $h \leftarrow_{\$} \{0, 1\}^{512}$	// All
2: $\beta \leftarrow h[0 : 32]$	// 1+	2: $v \leftarrow \perp$	// All
3: $A' \leftarrow \beta \cdot A$	// 1+	3: <b>if</b> $\nexists q', h' : (q', h', A') \in T_1 \wedge A' \neq A$	// 2+
4: <b>if</b> $\beta \equiv 0 \pmod{L}$	// 1+	4: $U \leftarrow U \cup \{A'\}$	// 2+
5: <b>Abort Game</b>	// 1+	5: <b>else</b>	// 3+
6: <b>if</b> $A' \in U$	// 2+	6: <b>if</b> $A' = A$	// 3+
7: <b>Abort Game</b>	// 2+	7: $\beta \leftarrow 1$	// 3+
8: <b>if</b> $bk = sk \wedge$ query is from $\mathcal{A}$	// 4+	8: <b>else</b> $\exists q', h' : (q', h', A') \in T_1$	// 3+
9: <b>Abort Game</b>	// 4+	9: $\beta \leftarrow h'[0 : 32]$	// 3+
10: $T_1 \leftarrow T_1 \cup \{(q, h, A')\}$	// All	10: $v \leftarrow_{\$} \{0, \dots, L-1\}$	// 3+
11: <b>return</b> $h$	// All	11: $h_{\mathcal{B}} \leftarrow H_{\mathcal{B}}(R + vA', A, m)$	// 3+
<hr/>		12: $z \leftarrow \beta^{-1}h_{\mathcal{B}} + v \pmod{L}$	// 3+
$H_{\mathcal{A}}(q = pre\ m, pre\ pre'\ m)$		13: $k \leftarrow_{\$} \{0, 1, \dots, d\}$	// 3+
1: $h \leftarrow_{\$} \{0, 1\}^{512}$	// All	14: $h \leftarrow z + kL$	// 3+
2: $pre^* \leftarrow H_{\mathcal{A}}(sk)[32 : 64]$	// 5+	15: <b>if</b> $h \geq 2^{512}$	// 3+
3: <b>if</b> $pre = pre^*$	// 5+	16: <b>Abort Game</b>	// 3+
4: <b>Abort Game</b>	// 5+	17: $T_2 \leftarrow T_2 \cup \{(q, h, v)\}$	// All
5: $T_3 \leftarrow T_3 \cup \{(q, h)\}$	// All	18: <b>return</b> $h$	// All
6: <b>return</b> $h$	// All		

Figure 6: Random oracle changes across games.

$\text{SignOracle}_0(bk, m)$	$\text{SignOracle}_6(bk, m)$	$\text{SignOracle}_7(bk, m)$
1: $x, A, pre \leftarrow \text{InnerKeyGen}(sk)$	<b>if</b> $bk = \perp$	$(R, s) \leftarrow \text{Sign}(m)$
2: <b>if</b> $bk = \perp$	$\beta \leftarrow 1$	<b>if</b> $bk = \perp$
3: $\beta \leftarrow 1$	<b>else</b>	$\beta \leftarrow 1$
4: $r \leftarrow H_{\mathcal{A}}(pre\ m)$	$\beta \leftarrow H_{\mathcal{A}}(bk)[0 : 32]$	<b>else</b>
5: <b>else</b>	$x' \leftarrow x \cdot \beta \pmod{L}$	$\beta \leftarrow H_{\mathcal{A}}(bk)[0 : 32]$
6: $\beta\ pre' \leftarrow H_{\mathcal{A}}(bk)$	$A' \leftarrow \beta \cdot A$	$A' \leftarrow \beta \cdot A$
7: $r \leftarrow H_{\mathcal{A}}(pre\ pre'\ m)$	$r \leftarrow_{\$} \{0, 1\}^{512}$	$v \leftarrow_{\$} \{0, \dots, d\}$
8: $x' \leftarrow x \cdot \beta \pmod{L}$	<b>if</b> $r \geq d \cdot L$	$z \leftarrow \beta^{-1}H_{\mathcal{B}}(R, A, m) + v \pmod{L}$
9: $A' \leftarrow \beta \cdot A$	<b>Abort Game</b>	$k \leftarrow_{\$} \{0, \dots, d\}$
10: $R \leftarrow r \cdot G$	$R \leftarrow r \cdot G$	$h \leftarrow z + kL$
11: $k \leftarrow H_{\mathcal{A}}(R, A', m)$	$k \leftarrow H_{\mathcal{A}}(R, A', m)$	$R' \leftarrow R - vA'$
12: $s \leftarrow r + kx' \pmod{L}$	$s \leftarrow r + kx' \pmod{L}$	<b>if</b> $(R', A', m) \in T_2$ or $h \geq 2^{512}$
13: <b>return</b> $(R, s)$	<b>return</b> $(R, s)$	<b>Abort Game</b>
14:		$T_2 \leftarrow T_2 \cup \{(R', A', m), h, v\}$
15:		<b>return</b> $(R - vA', s)$
16:		

Figure 7: Signing oracle changes across games.

- (4) We can do this by sampling  $k \leftarrow \{0, 1, \dots, d\}$  where  $d$  is the smallest integer such that  $L \cdot (d+1) > 2^{512}$ , and setting  $y \leftarrow z + k \cdot L$ . If  $y \geq 2^{512}$ , return  $\perp$  and abort the reduction.
- (5) Otherwise, record  $(q, y, v)$  in  $T_2$  and return  $y$ .

To consider how much  $G_3$  affects the probability the adversary wins, we must consider the distribution of the resulting  $y$  and the probability that we abort. Note that for each  $y \in \{0, \dots, 2^{512} - 1\}$ , there is a unique  $z$  and  $k$  such that  $z + k \cdot L = y$ . Furthermore, since

$v$  is a uniform and independent value mod  $L$  for each query the adversary makes, so is  $z$ , regardless of what  $\beta$  and  $h$  are. As well,  $k$  is uniform. So as long as we don't abort, the resulting  $y$  is uniform and independent of anything else in the game, and therefore the distribution of outputs to the random oracle is indistinguishable. Therefore all we need to consider is the probability that the game is aborted. We then consider the chances that  $z + k \cdot L \geq 2^{512}$  and show that it is negligible. Because  $d$  is the smallest integer such that  $(d + 1) \cdot L > 2^{512}$ , if  $k \leq d - 1$ , then  $(k + 1) \cdot L \leq 2^{512}$ . As  $z \leq L - 1$ ,  $z + k \cdot L < (k + 1) \cdot L \leq 2^{512}$ . So the only way for  $z + k \cdot L \geq 2^{512}$  is for  $k = d$ , which happens with probability  $1/(d+1)$ . As  $d > 2^{259}$ , this is cryptographically negligible. Therefore  $|\Pr_{G_3}[\mathcal{A} \text{ wins}] - \Pr_{G_2}[\mathcal{A} \text{ wins}]| \leq \frac{1}{d+1} < \frac{1}{2^{259}}$ .

In game  $G_4$  we abort if the adversary ever guesses the secret key, that is, if they ever query  $sk$  to  $H_{\mathcal{A}}$ . The adversary has no information about  $sk$  and so their only option is to guess, meaning that  $|\Pr_{G_4}[\mathcal{A} \text{ wins}] - \Pr_{G_3}[\mathcal{A} \text{ wins}]| \leq \frac{q_H}{2^{256}}$ .

In game  $G_5$  we abort if the adversary is able to guess at the prefix used to generate the value  $r$ . Recall that two prefixes are used to generate the  $r$  value when `BLKEYSIGN` is invoked:  $pre$  and  $pre'$ , generated from  $sk$  and  $sk'$  respectively. In this game, we abort if the adversary ever queries  $pre$  as a prefix to  $H_{\mathcal{A}}$ . Since we already abort if the adversary queries  $sk$  to the random oracle, the adversary has no information on  $pre$ , and thus their probability of success can be bounded by their ability to guess  $pre$ . So,  $|\Pr_{G_5}[\mathcal{A} \text{ wins}] - \Pr_{G_4}[\mathcal{A} \text{ wins}]| \leq \frac{q_H}{2^{256}}$ .

In game  $G_6$  we modify the signing oracle to slightly change the distribution of  $r$ . Rather than using  $H_{\mathcal{A}}$  to generate  $r$  we sample  $r \leftarrow \{0, 1\}^{512}$  and then abort if  $r \geq d \cdot L$ . The point of this is that it ensures that the distribution of  $r$  is now perfectly uniform modulo  $L$ . Since we have already ensured that the adversary does not query the proper prefix to learn what  $r$  should be, we only need consider the probability that we abort. In this case that probability can be computed exactly as  $\frac{2^{512} \pmod{L}}{2^{512}}$ , which will simply bound by  $\frac{L}{2^{512}} < \frac{1}{2^{259}}$ . Thus  $|\Pr_{G_6}[\mathcal{A} \text{ wins}] - \Pr_{G_5}[\mathcal{A} \text{ wins}]| \leq \frac{1}{2^{259}}$ .

Finally in game  $G_7$  we simulate the signing oracle rather than honestly generating signatures. When the adversary makes a signing query  $(bk, m)$ , perform the following:

- (1) If the an identical signing query has been made before, provide the same response.
- (2) Otherwise, we first query the (plain) signing oracle on the message  $m$ . The result is a signature  $(R, s)$ , which verifies with respect to  $H_{\mathcal{B}}$  and  $A$ . In other words, we have that  $sG = R + H_{\mathcal{B}}(R, A, m)A$ .
- (3) Compute  $\beta \leftarrow H_{\mathcal{A}}(sk')[0 : 32]$  and  $A' \leftarrow \beta A$ . Or, if  $bk = \perp$  (the adversary wants a signature under the unblinded public key) set  $A' \leftarrow A$  and  $\beta \leftarrow 1$ .
- (4) Sample a random  $v \leftarrow \{0, \dots, L - 1\}$ .
- (5) Program  $H_{\mathcal{A}}$  so that  $H_{\mathcal{A}}(R - vA', A', m) \equiv \beta^{-1}H_{\mathcal{B}}(R, A, m) + v \pmod{L}$ . That is, compute  $z \leftarrow \beta^{-1}H_{\mathcal{B}}(R, A, m) + v \pmod{L}$ , then sample a  $k \leftarrow \{0, \dots, d\}$ , compute  $h \leftarrow z + k \cdot L$  and add  $((R - vA', A', m), h, v)$  to  $T_2$ . If the query  $(R - vA', A', m)$  already appears in table  $T_2$  or  $h \geq 2^{512}$ , then abort the reduction.
- (6) Return signature  $(R - vA', s)$ .

The translation discussed in the proof summary (Equation 1) establishes that this is indeed a valid signature for  $\mathcal{A}$ . Furthermore, our reprogramming of  $H_{\mathcal{A}}$  maintains the translation between valid signatures. There are only two small ways in which an adversary could detect that we are not performing an honest signing execution. First is that the  $R - vA'$  sent to the adversary is generated in an atypical way, instead of being calculated from  $H_{\mathcal{A}}((pre || pre'), m)G$ . But we already abort the reduction if the adversary ever makes a query to  $H_{\mathcal{A}}$  with  $pre$  as a prefix, so this can only be detected by the distribution of the signature. But note that the point  $R - vA'$  still has the proper distribution. Since  $A'$  is a point of order  $L$ , by multiplying by  $v$  we get a uniformly random point of order  $L$  (or with probability  $1/L$ , the identity point). This is also true of the  $R$  point in game  $G_6$ . Thus there is no difference in the distribution of the “ $R$ ” value of the signature, and as there is only one valid  $s$  value for any given  $R$ , the distribution of the entire signature is correct.

The second possibility is that we are forced to abort the signing procedure because  $(R - vA', A', m)$  was already queried to  $H_{\mathcal{A}}$ , and thus the oracle cannot be programmed. But note that since  $A' \neq 0$  (the identity point), then we again have that  $vA'$  is a uniform point, so that the input has a high degree of entropy, so that it is unlikely that the adversary could possibly have queried it before. So this means that  $R - vA'$  can take on any of  $L$  values, and so the probability that it has already been queried is at most  $q_H/L$ , making the probability that the adversary is able to cause this to happen negligible for each signing query.

Finally, there is the probability that the resulting  $h$  is larger than  $2^{512}$ . Just as in the game hop between games  $G_2$  and  $G_3$ , this can be bounded by  $1/(d + 1) < 1/2^{259}$ . So we can bound the difference between games  $G_5$  and  $G_6$  by  $|\Pr_{G_7}[\mathcal{A} \text{ wins}] - \Pr_{G_6}[\mathcal{A} \text{ wins}]| \leq \frac{q_S \cdot q_H}{L} + \frac{1}{2^{259}}$ .

**Signature Extraction.** Finally, we need to establish that if an adversary submits a forgery, this can be translated to a forgery for with respect to  $A$  and the random oracle  $H_{\mathcal{B}}$ . Without loss of generality, we can assume that the adversary has verified their own forgery. Therefore, they have submitted a  $m^*, bk^*, (R^*, s^*)$  such that if

$$A'^* \leftarrow \text{Ed25519.BIPubKey}(A, bk),$$

then  $\text{Ed25519.Verify}(A'^*, m^*, (R^*, s^*))$  accepts and

$$(bk^*, m^*, (R^*, s^*)) \notin \Sigma.$$

For the signature to verify, we have that

$$s^*G = R^* + H_{\mathcal{A}}(R^*, A'^*, m^*)A'^*.$$

At this point, we can take  $q = (R^*, A'^*, m^*)$  to look up in table  $T_2$ . Since the signature has been verified, it must appear in the table. There are two possibilities. Either we find a response  $y$  and value  $v$ , or we find just the response and  $\perp$ . This second case cannot happen, as it would mean that  $A'^*$  was added to the list of unexplained public keys, and then later was constructed as part of a blinding.

Thus we can safely consider the case that when the signature is submitted, we find in table  $T_2$  a response  $y$  and values  $\beta$  and  $v$ . Whether this entry was added by result of a hash or a signing query, we know that  $y = \beta^{-1}H_{\mathcal{B}}(R + vA'^*, A, m) + v$ . From equation 1, we have that this means that  $(s^*, R + vA'^*)$  is a valid signature. The

remaining question is whether it is a valid *forgery*, that is whether  $(m, (s^*, R + \nu A'^*))$  is a new message-signature pair.

There are two possibilities: either  $m$  was already submitted as a signing query in  $\mathcal{B}$ 's strong unforgeability game or it was not. If it was not, then clearly  $m$  is a valid forgery. If it was, then we need to show that the signature is new. The potential problem is this: the adversary  $\mathcal{A}$  submits a signing query  $(bk_1, m_1)$  to the BLKEYSIGN oracle. This is resolved into a valid signature as described above, meaning that first  $m_1$  is sent to the signing oracle, resulting in a signature  $(s_1, R_1)$ , valid with respect to  $A$  and  $H_{\mathcal{B}}$ . This is then translated into a signature  $(s_2, R_2)$  that is valid with respect to  $A'_1 = \text{Ed25519.BIPubKey}(A, bk_1)$  and the oracle  $H_{\mathcal{A}}$ . When the adversary submits the forgery  $(m^*, bk^*, (s^*, R^*))$ , we have that the tuple  $(bk, s^*, R^*) \neq (bk_1, s_2, R_2)$ , making it a valid forgery, but when translated into a signature with respect to  $A$  and  $H_{\mathcal{B}}$  as described above, the result is  $(s_1, R_1)$ . In other words, breaking the strong unforgeability means finding a sort of 'collision' in our method of translating between valid signatures, one coming from an honest signing query, the other not.

The reason that this should not happen is because the random choice of  $\nu$  will make it negligibly likely that such a collision occurs. When the adversary submits a their forgery query  $(R^*, A', m^*)$ , this will be translated by a random  $\nu$  into  $(R^* - \nu A', A, m^*)$ . The chance that this actually matches a previous signature  $(R, A, m^*)$  is just the chance that  $R^* - \nu A' = R$ , which is a straightforward  $1/L$  chance. So each time the adversary submits a hash query, the chance that it matches one of the previous signature queries is  $q_S/L$ , resulting in an overall chance of this happening of  $q_H q_S/L$ . As a result, we can calculate that  $|\Pr[\text{Forgery extracted}] - \Pr_{G_6}[\mathcal{A} \text{ wins}]| \leq \frac{q_H q_S}{L}$ .

Putting the differences between the subsequent games and the ability to extract a forgery provides the expected bound.  $\square$

## 5 SECURITY ANALYSIS OF ECDSA WITH KEY BLINDING

In this section, we analyze the security of the rate-limited issuance protocol components build on ECDSA. As we did in 4, we focus on proving two properties: (1) unlinkability and unforgeability of ECDSA with key key blinding, and (2) unforgeability and pseudo-randomness of BK-PRF built on ECDSA.

### 5.1 Unlinkability

We prove unlinkability directly, rather than relying on the framework in Appendix B. This is because of difficulties in establishing the signing with oracle reprogramming property for ECDSA signatures with key-blinding. While simulating signatures by programming the random oracle is possible with ECDSA (as proven by Fersch, Kiltz, and Poettering [13]), there are technical problems when one tries to adapt this technique to the key-blinding case. This is primarily because the random oracle input that needs to be programmed is simply the message  $m$ , which has no entropy. As a result, we cannot program the random oracle at the time of simulating a signature, and instead we need to simulate the signature when the random oracle is first queried (and then if  $m$  is queried to the signing oracle later, respond consistently). But this in turn causes issues because we do not know which  $bpk$  value to simulate the signature for.

Because of this, we take a different and more direct approach to proving unlinkability. We note that this approach, while less generic than the framework in Appendix B, is actually much simpler, and likely also works for Ed25519. We prove the following.

**THEOREM 5.1 (ECDSA INDEPENDENT BLINDING).** *For any probabilistic polynomial time adversary  $\mathcal{A}$ , we have that  $\text{Adv}_{\mathcal{A}, \text{ECDSA}}^{\text{IndBlind}} \leq q_H/2^{256}$ .*

**PROOF.** We proceed by a game hopping proof. The original game  $G_0$  is the unlinkability game in Figure 2, instantiated with ECDSA. Then, in game  $G_1$  we replace the inner workings of BLPUBLICKEY by first sampling  $bk$ , but then rather than blinding the identity public we, we simply sample a new key pair from  $\text{ECDSA.KeyGen}()$ , and adding the resulting  $(pk, sk)$  to  $\Sigma$  and returning  $pk$ . To match this change, we also modify the BLSign oracle. So long as  $bpk \neq bpk^*$ , when a  $(m, pk)$  is submitted, we check for  $(pk, sk) \in \Sigma$  and simply return  $\text{ECDSA.Sign}(sk, m)$ .

We can do this because in ECDSA, for any keypair  $(pk_S, sk_S)$  the distribution of  $(\beta \cdot pk_S, \beta \cdot sk_S)$  is identical to that of a freshly sampled  $(pk, sk)$  (over the randomness in sampling  $\beta$ ). So long as the adversary does not query  $H_2S(bk)$  there is no way to distinguish that the blinding key is not being constructed properly. As each  $bk$  is a uniform 32 bytes, the probability that the adversary is able to query a  $bk$  in  $q_H$  random oracle queries is a straightforward  $q_H/2^{256}$ , which is thus the difference between the adversary's advantage in games  $G_0$  and  $G_1$ .

For game  $G_2$ , we delay the sampling of the identity keypair  $(pk_S, sk_S)$  to when they are needed. Since they are no longer used in the BLPUBLICKEY or in the BLKEYSIGN until the CHALLENGE oracle is called. The first time the identity keypair is used is then in the  $b = 1$  branch of execution of the CHALLENGE oracle, meaning we would move  $sk_S, pk_S \leftarrow \text{KeyGen}()$  to between line 6 and 7 of CHALLENGE. This does not cause a difference in the adversary's advantage whatsoever, and is purely a syntactic change.

But we can now observe in game  $G_2$  that the two execution branches depending on the bit  $b$  are entirely identical. Thus, the adversary has no advantage in determining the bit  $b$ , and we are able to derive the expected bound.  $\square$

### 5.2 Unforgeability

The EUF-CMA security game is that of [17, Def. 1] modified so that the signing oracle takes a message  $m$  and a blinding parameter  $bk$ . Similarly, the adversary must produce a signature on a message  $m'$  under a specific blinding parameter  $bk' \in \{0, 1\}^* \cup \{\perp\}$ , and the adversary wins if the signature is valid and the pair  $(m', bk')$  was not given to the signing oracle. We define  $\text{Adv}_{\text{cma}}^{\text{ecgmm}}[\mathcal{A}, S_{\text{ecdsa}}]$  to be the advantage of  $\mathcal{A}$  in winning this modified game in the elliptic-curve generic-group model.

We prove the following theorem, using the properties on hash functions as defined in Appendix C.

**THEOREM 5.2.** *Let  $\mathcal{A}$  be an adversary attacking  $S_{\text{ecdsa}}$  defined in Fig. 5, that makes at most  $N$  signing or group queries. Then there exists adversaries  $\mathcal{B}_{Ia}, \mathcal{B}_{Ib}, \mathcal{B}_{Ic}, \mathcal{B}_{IIa}, \mathcal{B}_{IIb}, \mathcal{B}_{III}$ , whose running*

times are essentially the same as  $\mathcal{A}$ , such that

$$\begin{aligned} \text{Adv}_{\text{cma}}^{\text{ecggm}}[\mathcal{A}, S_{\text{ecdsa}}] &\leq \text{Adv}_{\text{collision}}[\mathcal{B}_{Ia}, \text{Hash}] \\ &+ \text{Adv}_{1\text{prod}}[\mathcal{B}_{Ib}, \text{Hash}, \text{H2S}] \\ &+ \text{Adv}_{2\text{prod}}[\mathcal{B}_{Ic}, \text{Hash}, \text{H2S}] \\ &+ (4 + o(1))N\text{Adv}_{\text{rpr}}[\mathcal{B}_{IIa}, \text{Hash}] \\ &+ (4 + o(1))N\text{Adv}_{\text{quot}}[\mathcal{B}_{IIb}, \text{Hash}, \text{H2S}] \\ &+ \text{Adv}_{\text{zpr}}[\mathcal{B}_{III}, \text{Hash}]. \end{aligned}$$

We deduce the following corollary.

**COROLLARY 5.3.** *If Hash and H2S are modeled as random oracle where  $Q_i$  for  $i \in \{1, 2\}$  is a bound on the number of queries to  $H_i$  and  $N$  a bound on the number of signing or group queries made by  $\mathcal{A}$ , then*

$$\text{Adv}_{\text{cma}}^{\text{ecggm,ro}}[\mathcal{A}, S_{\text{ecdsa}}] \leq O\left(\max(N, Q_1 Q_2) \cdot \frac{Q_1 Q_2}{q}\right).$$

Note that all the transformations from [17, Lemma 1] defining a lazy symbolic simulator hold, where to process a signing query  $(h, bk)$ , the symbolic simulator runs the same algorithm but with  $\text{InnerBlind}(bk) \cdot \mathbf{D}$  in place of  $\mathbf{D}$ . Denote  $(m', (s', t'), bk')$  the adversary forgery and denote  $h' = \text{Hash}(m')$ ,  $\beta' = \text{InnerBlind}(bk')$  and

$$\mathcal{R}' = (s')^{-1}h'\mathcal{G} + (s')^{-1}t'\beta'\mathcal{D}$$

computed during verification.

Similarly to [17, Sec. 5], let's define the following types of forgers.

**Type I.**  $\mathcal{R}' = \pm\mathcal{R}$  for some  $\mathcal{R}$  computed by the signing oracle.

**Type II.**  $\mathcal{R}' \neq \pm\mathcal{R}$  for any  $\mathcal{R}$  computed by the signing oracle and  $h' \neq 0$

**Type III.** Neither Type I nor Type II.

**Type I forgeries.** First, consider a type I forger where  $\mathcal{R}' = \pm\mathcal{R}$  for some  $\mathcal{R}$  computed by the signing oracle (which must be unique). There exists  $s, t, h$ , and  $bk$  such that

$$(s')^{-1}(h' + t'\beta'\mathbf{D}) = \pm s^{-1}(h + t\beta\mathbf{D}), \quad \text{and} \quad t = t'$$

where  $\beta = \text{InnerBlind}(bk)$ . Hence, there exists  $\mu \in \{-1, 1\}$  such that

$$(s')^{-1}(h' + t'\beta'\mathbf{D}) = \mu s^{-1}(h + t\beta\mathbf{D})$$

which yields

$$(s')^{-1}h' = \mu s^{-1}h \quad \text{and} \quad (s')^{-1}t'\beta' = \mu s^{-1}t\beta,$$

and therefore

$$h'/\beta' = h/\beta.$$

If  $\beta' = \beta$  (Type Ia forgeries), then  $h' = h$  and this implies a collision on the hash function Hash. If  $bk = \perp$  or  $bk' = \perp$  (Type Ib forgeries), then such a forger can be used to solve the “1prod” property of Hash and H2S. If  $bk, bk' \neq \perp$  (Type Ic forgeries), then such a forger can be used to solve the “2prod” property of Hash and H2S.

We have shown that,

- if  $\mathcal{A}$  is an efficient adversary which produces a Type Ia forgery with probability  $\epsilon_{Ia}$ , there exists an efficient adversary  $\mathcal{B}_{Ia}$  such that  $\text{Adv}_{\text{collision}}[\mathcal{B}_{Ia}, \text{Hash}] \geq \epsilon_{Ia}$ .

- if  $\mathcal{A}$  is an efficient adversary which produces a Type Ib forgery with probability  $\epsilon_{Ib}$ , there exists an efficient adversary  $\mathcal{B}_{Ib}$  such that  $\text{Adv}_{1\text{prod}}[\mathcal{B}_{Ib}, \text{Hash}, \text{H2S}] \geq \epsilon_{Ib}$ .
- if  $\mathcal{A}$  is an efficient adversary which produces a Type Ic forgery with probability  $\epsilon_{Ic}$ , there exists an efficient adversary  $\mathcal{B}_{Ic}$  such that  $\text{Adv}_{2\text{prod}}[\mathcal{B}_{Ic}, \text{Hash}, \text{H2S}] \geq \epsilon_{Ic}$ .

### Type II forgeries.

Next, consider a type II forger where  $\mathcal{R}' \neq \pm\mathcal{R}$  for any  $\mathcal{R}$  computed by the signing oracle. Suppose

$$\pi^{-1}(\mathcal{R}') = a + b\mathbf{D}.$$

By the verification equation, we have that  $\pi^{-1}(\mathcal{R}') = (s')^{-1}(h' + t'\beta'\mathbf{D})$ , hence

$$a = (s')^{-1}h' \quad \text{and} \quad b = (s')^{-1}t'\beta'.$$

Since  $h' \neq 0$ , then  $a \neq 0$  and the previous equations imply that  $t' = (b/a) \cdot h'/\beta'$ . Note that the group element  $\mathcal{R}'$  must have been generated at random by some group oracle query made directly by the adversary (since it was not generated during a signature query). Since the coefficients  $a, b$  were already determined before this query, the values of  $\mathcal{R}'$  is independent of these coefficients.

If  $bk' = \perp$  (Type IIa forgeries), then  $t' = (b/a) \cdot h'$  and we are back exactly to [17, Sec. 5.3, Type II]. Henceforth, we can use such a forger to breaking the random-preimage resistance of Hash. Otherwise (Type IIb forgeries), we use such a forger to break the quotient property of Hash and H2S by choosing  $e = a/b$  and the value  $t$  would correspond to  $t'$ .

We have shown that,

- if  $\mathcal{A}$  is an efficient adversary that makes at most  $N$  signing or group queries and which produces a Type IIa forgery with probability  $\epsilon_{IIa}$ , there exists an efficient adversary  $\mathcal{B}_{IIa}$  such that  $\text{Adv}_{\text{rpr}}[\mathcal{B}_{IIa}, \text{Hash}] \geq (1/4 + o(1))\epsilon_{IIa}/N$ ,
- if  $\mathcal{A}$  is an efficient adversary that makes at most  $N$  signing or group queries and which produces a Type IIb forgery with probability  $\epsilon_{IIb}$ , there exists an efficient adversary  $\mathcal{B}_{IIb}$  such that  $\text{Adv}_{\text{quot}}[\mathcal{B}_{IIb}, \text{Hash}, \text{H2S}] \geq (1/4 + o(1))\epsilon_{IIb}/N$ .

**Type III forgeries.** Those forgeries directly gives us a forgery with  $h' = 0$ , which immediately yields an adversary can that break the zero preimage resistance of Hash.

## 6 BENCHMARKS

In this section we provide benchmarks that capture the amount of overhead the EdDSA and ECDSA signature schemes with key blinding add on top of the base schemes. In particular, we benchmark all functions necessary for key generation, public key blinding, public key unblinding, and blind key signing. We configure ECDSA over P-384 and SHA-384<sup>3</sup>. We fix the length of the  $\text{ctx}$  and  $m$  to 32 random bytes for each operation, since this is a reasonable length for most known applications. We do not vary the context length as the base signature scheme does not support context strings, nor do we vary the message length since messages are input to hash functions and consequently have significantly less cost compared to public key operations.

<sup>3</sup>We chose P-384 as larger groups are necessary to deal with the semi-static DH oracle attacks in Privacy Pass.

We ran the performance benchmarks on a laptop computer running macOS 12.6 with a 2.6 Ghz 6-Core Intel Core i7 CPU and with 32 GB of RAM. The results for EdDSA and ECDSA are shown in Table 1. With a fixed message length of 32 bytes, blind key signing is 200.4% slower than the base signing scheme for Ed25519, and blind key signing is 315.5% slower than the base signing scheme for ECDSA. These results are not surprising given that blinding the signing key, at a minimum, doubles the number of private key operations required for each scheme.

**Table 1: Computation costs for Ed25519 and ECDSA signing with key blinding.**

Operation	Time (ns/op)	
	Ed25519	ECDSA
BlGen	1258	339555
BlPubKey	87351	1025784
UnblindPublicKey	84238	1005225
BlSign	114192	1490511
Sign	32492	350035

## 7 RELATED WORK

This work is not the first to study signature schemes with key blinding. Hopper analyzed a variant based on EdDSA specifically in the context of Tor [19]. However, this work is inherently tied to the context that Tor uses for key blinding and is not generally applicable. Beyond applicability, the unforgeability proof is non-tight. It amounts to the adversary essentially guessing the random oracle query that is going to result in a forgery. In contrast, our proofs are tight. Finally, the unforgeability security definition does not consider it a break if a given  $(m^*, bk^*)$  pair was queried before, but  $\sigma^*$  is new (i.e., strong unforgeability). In contrast, our definitions cover this stronger notion of unforgeability.

Fleischhacker et al. [15] also study this topic. However, their work does not consider unlinkability of signatures with key blinding. In particular, their unforgeability security definition considers a tuple  $(m^*, bk^*, \sigma^*)$  a forgery if the message has never been submitted to the signing procedure before. In contrast, for EdDSA we consider it a forgery if any of the following holds:  $m^*$  has never been submitted, or it was submitted with respect to a different  $bk^*$ , or  $(m^*, bk^*)$  was submitted but the signature that resulted was different (i.e., strong unforgeability). As such, our definition is stronger, especially since signatures that support key blinding are often vulnerable to malleability attacks.

Wahby et al. [24] also study ECDSA and EdDSA signature variants with a key blinding like property, yet study different properties. Unforgeability is similarly defined, asking the adversary to produce a valid signature over any  $m^*$  with respect to a *specific*  $sk$  given access to any other message and signature pairs. However, their work does not explicitly cover unlinkability. Instead, it covers a notion of *anonymity* that is specific to the airdrop use case. Anonymity is defined with respect to a single signing transcript, rather than arbitrarily many transcripts, and also permits some quantified amount

of leakage about the signing key pair. In contrast, our unlinkability definition does not permit any leakage and is not constrained to just one transaction.

Morita et al. [21] showed that ECDSA with naive multiplicative key blinding, i.e., without the  $H2S$  function to map  $bk$  to a scalar that blinds the private and public key, was vulnerable to forgeries. The construction in Section 3 resolve this by removing the linear relationship between related keys with  $H2S$ .

Groth and Shoup proved security of ECDSA with *additive* key derivation in [16]. The concrete construction in this work uses multiplicative blinding, in particular because certain applications of this scheme want repeated blindings of the same private and public key, i.e.,  $bpk' \leftarrow \text{BIPubKey}(\text{BIPubKey}(pk, bk, ctx), bk', ctx)$ .

## 8 CONCLUSION

In this work we present a formal analysis of signature schemes with key blinding, a primitive that’s deployed in practice and being standardized for the purposes of more widespread use. We prove that the EdDSA and ECDSA variants being standardized meet the desired security properties. Our experimental results demonstrate that they add modest amount of overhead compared to the base signature scheme. It is an open question to build such primitives with post quantum security. Future work should explore this topic, especially as this primitive finds more widespread use in practice.

## REFERENCES

- [1] 2021. Web Authentication: An API for accessing Public Key Credentials Level 2. <https://www.w3.org/TR/webauthn-2/>.
- [2] Apple. 2022. Replace CAPTCHAs with Private Access Tokens. <https://developer.apple.com/videos/play/wwdc2022/10077/>.
- [3] Apple. 2023. <https://register.apple.com/pat>.
- [4] Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose. 2005. DNS Security Introduction and Requirements. RFC 4033. <https://doi.org/10.17487/RFC4033>
- [5] AB Association et al. 2005. Public key cryptography for the financial services industry: The elliptic curve digital signature algorithm (ECDSA). *Amer. Nat. Standards Inst* (2005), 62–1998.
- [6] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. 2018. Privacy Pass: Bypassing Internet Challenges Anonymously. *Proc. Priv. Enhancing Technol.* 2018, 3 (2018), 164–180.
- [7] Alex Davidson, Jana Iyengar, and Christopher A. Wood. 2023. *The Privacy Pass Architecture*. Internet-Draft draft-ietf-privacypass-architecture-10. Internet Engineering Task Force. <https://datatracker.ietf.org/doc/draft-ietf-privacypass-architecture/10/> Work in Progress.
- [8] Frank Denis, Edward Eaton, Tancrede Lepoint, and Christopher A. Wood. 2023. *Key Blinding for Signature Schemes*. Internet-Draft draft-irtf-cfrg-signature-key-blinding-03. Internet Engineering Task Force. <https://datatracker.ietf.org/doc/draft-irtf-cfrg-signature-key-blinding/03/> Work in Progress.
- [9] Frank Denis, Frederic Jacobs, and Christopher A. Wood. 2022. *RSA Blind Signatures*. Internet-Draft draft-irtf-cfrg-rsa-blind-signatures-07. Internet Engineering Task Force. <https://datatracker.ietf.org/doc/draft-irtf-cfrg-rsa-blind-signatures/07/> Work in Progress.
- [10] Tim Dierks and Christopher Allen. 1999. *RFC 2246 - The TLS Protocol Version 1.0*. Internet Activities Board.
- [11] Edward Eaton, Douglas Stebila, and Roy Stracovsky. 2021. Post-quantum Key-Blinding for Authentication in Anonymity Networks. In *LATINCRYPT 2021 (LNCS, Vol. 12912)*, Patrick Longa and Carla Ràfols (Eds.). Springer, Heidelberg, 67–87. [https://doi.org/10.1007/978-3-030-88238-9\\_4](https://doi.org/10.1007/978-3-030-88238-9_4)
- [12] Armando Faz-Hernandez, Sam Scott, Nick Sullivan, Riad S. Wahby, and Christopher A. Wood. 2022. *Hashing to Elliptic Curves*. Internet-Draft draft-irtf-cfrg-hash-to-curve-16. Internet Engineering Task Force. <https://datatracker.ietf.org/doc/draft-irtf-cfrg-hash-to-curve/16/> Work in Progress.
- [13] Manuel Fersch, Eike Kiltz, and Bertram Poettering. 2017. On the One-Per-Message Unforgeability of (EC)DSA and Its Variants. In *TCC 2017, Part II (LNCS, Vol. 10678)*, Yael Kalai and Leonid Reyzin (Eds.). Springer, Heidelberg, 519–534. [https://doi.org/10.1007/978-3-319-70503-3\\_17](https://doi.org/10.1007/978-3-319-70503-3_17)
- [14] Nils Fleischhacker, Johannes Krupp, Giulio Malavolta, Jonas Schneider, Dominique Schröder, and Mark Simkin. 2016. Efficient Unlinkable Sanitizable

- Signatures from Signatures with Re-randomizable Keys. In *PKC 2016, Part 1 (LNCS, Vol. 9614)*, Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang (Eds.). Springer, Heidelberg, 301–330. [https://doi.org/10.1007/978-3-662-49384-7\\_12](https://doi.org/10.1007/978-3-662-49384-7_12)
- [15] Nils Fleischhacker, Johannes Krupp, Giulio Malavolta, Jonas Schneider, Dominique Schröder, and Mark Simkin. 2016. Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys. In *Public-Key Cryptography–PKC 2016: 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6–9, 2016, Proceedings, Part I*. Springer, 301–330.
- [16] Jens Groth and Victor Shoup. 2021. On the security of ECDSA with additive key derivation and presignatures. *Cryptology ePrint Archive*, Paper 2021/1330. <https://eprint.iacr.org/2021/1330> <https://eprint.iacr.org/2021/1330>.
- [17] Jens Groth and Victor Shoup. 2022. On the Security of ECDSA with Additive Key Derivation and Presignatures. In *EUROCRYPT 2022, Part I (LNCS, Vol. 13275)*, Orr Dunkelman and Stefan Dziembowski (Eds.). Springer, Heidelberg, 365–396. [https://doi.org/10.1007/978-3-031-06944-4\\_13](https://doi.org/10.1007/978-3-031-06944-4_13)
- [18] Scott Hendrickson, Jana Iyengar, Tommy Pauly, Steven Valdez, and Christopher A. Wood. 2022. *Rate-Limited Token Issuance Protocol*. Internet-Draft draft-ietf-privacypass-rate-limit-tokens-00. Internet Engineering Task Force. <https://datatracker.ietf.org/doc/draft-ietf-privacypass-rate-limit-tokens/00/> Work in Progress.
- [19] Nicholas Hopper. 2013. Proving Security of Tor’s Hidden Service Identity Blinding Protocol. <https://www-users.cs.umn.edu/~hoppernj/basic-proof.pdf>.
- [20] Simon Josefsson and Ilari Liusvaara. 2017. Edwards-Curve Digital Signature Algorithm (EdDSA). RFC 8032. <https://doi.org/10.17487/RFC8032>
- [21] Hiraku Morita, Jacob C. N. Schuldt, Takahiro Matsuda, Goichiro Hanaoka, and Tetsu Iwata. 2015. On the Security of the Schnorr Signature Scheme and DSA against Related-Key Attacks. *Cryptology ePrint Archive*, Paper 2015/1135. <https://eprint.iacr.org/2015/1135> <https://eprint.iacr.org/2015/1135>.
- [22] Hiraku Morita, Jacob C. N. Schuldt, Takahiro Matsuda, Goichiro Hanaoka, and Tetsu Iwata. 2016. On the Security of the Schnorr Signature Scheme and DSA Against Related-Key Attacks. In *ICISC 15 (LNCS, Vol. 9558)*, Soonhak Kwon and Aaram Yun (Eds.). Springer, Heidelberg, 20–35. [https://doi.org/10.1007/978-3-319-30840-1\\_2](https://doi.org/10.1007/978-3-319-30840-1_2)
- [23] The Tor Project, Inc. 2020. Tor Rendezvous Specification - Version 3. <https://gitweb.torproject.org/torspec.git/tree/rend-spec-v3.txt>.
- [24] Riad S Wahby, Dan Boneh, Christopher Jeffrey, and Joseph Poon. 2020. An airdrop that preserves recipient privacy. In *Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers 24*. Springer, 444–463.

## A CONFIGURATIONS OF KEY-BLINDING

In this section we discuss a few of the design choices that can be made when implementing signatures with key-blinding and the extent to which these decisions can impact the security of the scheme.

### A.1 One-way vs. Bidirectional blinding

A key design decision in defining a key-blinding scheme is whether the blinding procedure is intended to be one-way or not. Consider key-blinding as used in the rate-limited privacy pass token [18]. Here the public keys used in key-blinding are used in combination with a kind of DH-OPRF protocol. A requirement of this is that one of the participants in the protocol (the attester) needs to ‘unblind’ the blinded public key, taking the blinding key  $bk$ , using it to compute the scalar  $\beta$  and computing  $\beta^{-1} \cdot bpk$ .

A critical point here is that the blinding key  $bk$  is a *secret value*, unavailable to the adversary. Thus the unblinding process is only possible for trusted actors, and we do not need to worry about the unlinkability of the scheme.

Contrast this with key-blinding as used in Tor [23]. Here the blinding key  $bk$  is formed from entirely public parameters available to anyone operating within the network. Despite this, the intention is that seeing a blinded public key leaks no information about the identity public key. This can be accomplished by including the identity public key in the context with any other data.

By doing this, when observing a blinded public key, the mathematical scalar  $\beta$  used to blind the public key is unknown without the identity public key. As a result, the blinded public key cannot be unblinded unless you already have the identity public key, rendering the unblinded process useless.

Choosing whether blinding should be one-way or bidirectional is an important design for protocol designers using key-blinding. In general, unless unblinding is absolutely required by the protocol, one-way blinding should be used. This strengthens the unlinkability property, because the adversary can learn and even control the  $bk$  value without being able to learn anything about the identity public key.

### A.2 Clamping

Secret key clamping is a procedure used to tweak the output distribution of an Ed25519 secret key. Recall that in RFC8032, Ed25519 secret keys are a uniformly random 32 bytes. This secret key is then hashed with SHA512 to generate 64 bytes, the lower 32 of which are used for the secret scalar. These 32 bytes are then pruned, or clamped: the lowest three bits are set to 0, the highest bit is set to 0, and the second highest bit is set to 1. The result is interpreted as a little-endian integer to be multiplied by the standard generator. The cumulative effect is that if the first 256 bits of the SHA512 hash of the secret key are  $a_0, a_1, \dots, a_{255}$  then the secret scalar will be equal to  $\sum_{i=3}^{253} a_i \cdot 2^i + 2^{254}$ .

When a  $bk$  is used to generate a blinded public key, designers and implementers have a choice whether to perform the same clamping procedure to generate the scalar  $\beta$  before being applied to the identity public key. Recall that  $L$  is greater than  $2^{252}$  but less than  $2^{253}$ . The result of clamping is that  $2^{251}$  of the possible scalars (mod  $L$ ) are ‘hit’; less than half.

This means that when an identity key is blinded using a clamped scalar, only half of the resulting key space can be hit. Conversely, from a blinded public key only half of the key space is a potential identity key. This means that from an information theoretic perspective, a blinded public key causes a bit of entropy of the identity public key to be lost. After observing 252 or so blinded public keys, the identity public key is (information theoretically) determined.

This is unlikely to result in a practical attack. Computationally, eliminating a potential identity public key as a candidate based off of a blinded public key requires knowing if the the scalar that leads to that blinded key is a possible output of the hash function, an infeasible task. Nonetheless we assume for the rest of the paper that clamping is *not* used to generate the blinding scalar, as the resulting distribution is much closer to uniform. This matches the recommendation in [8].

### A.3 Inclusion of the (blinded) public key in the hash function

As discussed in Section 2, in the definition of (strong) unforgeability we have provided, a forgery  $(m^*, bk^*, \text{Sig}^*)$  is considered valid if the signature verifies and if  $\text{Sig}^*$  was not the response to a query  $(m^*, bk^*)$  to the signing oracle. This means that if an adversary submits a query  $(m^*, bk)$  and then is able to generate a signature on the same message but with respect to a different  $bk^*$ , the forgery is considered valid and the adversary has broken the scheme.

One reason security in this model is preferable is that naively constructed schemes can be insecure in this model due to related key attacks [22]. Consider an alternative form of Ed25519 blinding where the scalar  $\beta$  is incorporated *additively* instead of multiplicatively. That is, we calculate  $bpk = \beta \cdot G + pk = (\beta + sk) \cdot G$  instead of  $\beta \cdot pk$ . If  $bpk$  is not included in the hash function when generating  $k$  for a signature  $(R, s)$  on a message  $m$ , then for any other blinding factor  $\beta'$ ,  $(R, s - k \cdot \beta + k \cdot \beta')$  is a valid signature on  $m$  with respect to the blinded public key generated with  $\beta'$ .

A key point enabling these attacks is whether the blinded public key  $bpk$  is included in the hash function when hashing the message as part of signing (e.g., on line 6 of BLSign in Figure 4 or line 1 of BLSign in Figure 5). Adding  $bpk$  here ties the signature to the  $bk$  used for that signature and can prevent this kind of related key attack, as we will see when we prove the unforgeability of Ed25519 in Section 4.2. We leave it as an open question on whether the inclusion of the blinded public key (as well as bound checks on  $s$ ) is all that is necessary to make ECDSA unforgeable in the sense of Figure 3.

## B UNLINKABILITY WITH UNBLINDING

In this section we replicate the proof technique in [11] to establish unlinkability for key-blinding schemes with *bidirectional*, as opposed to just one-way blinding. We recall the first property that we will reduce unlinkability to.

*Definition B.1.* (Signing with Oracle Reprogramming) Let Sig be a signature scheme that relies on a random oracle  $H$ . We say that the signature scheme admits signing with oracle reprogramming if there exists a reprogrammed point extractor Ext and a forgery function Forge that takes in  $pk, m$  and returns a  $(y, \sigma)$  such that  $\text{Sig.Verify}^{H:\text{Ext}(\sigma, pk, m)} \rightarrow y(pk, m, \sigma) \rightarrow 1$ , where  $H : x \mapsto y$  denotes a random oracle modified so that  $H(x) = y$ .

For more details on this point extractor function and forgery function we refer to [11]. Whether the adversary is able to notice that signing queries are being simulated rather than honestly generated can be established through two distributions: that of the joint distribution of the signature and the output of the hash function, and the whether the distribution of the input to the hash function is sufficiently high to ensure that the adversary has not previously queried it on a point that then needs to be programmed.

*Definition B.2.* (Signature Distribution Change) Let Sig be a signature scheme defined with respect to a random oracle  $H$  that admits signing with oracle reprogramming. For a public key  $pk$  and a message  $m$ , we consider the adversary's ability to distinguish the distribution  $(y_f, \sigma_f) \leftarrow \text{Forge}(pk, m)$  from the distribution of  $(y_r, \sigma_r)$  where  $\sigma_r \leftarrow \text{Sig.Sign}(sk, m)$  and  $y_r = H(\text{Ext}(\sigma_r, pk, m))$  (i.e., the real output of the hash function on the input that would otherwise need to be programmed). We denote the  $L1$  distance between these distributions as  $\delta$ , that is,

$$\delta_{\text{Sig}} = \sum_{\sigma, y} |\Pr[\sigma_r = \sigma, y_r = y] - \Pr[\sigma_f = \sigma, y_f = y]| \quad (3)$$

*Definition B.3.* (Programmed point min-entropy) Let Sig be a signature scheme with respect to a random oracle  $H$  that admits signing with oracle reprogramming. Let  $h_{\text{min, Sig}}$  denote the min-entropy of  $\text{Ext}(\sigma, pk, m)$ , where  $(y, \sigma) \leftarrow \text{Forge}(pk, m)$ .

Following the proof technique in [11], we first reduce the ability of an adversary to compromise the unlinkability property to an adversary who makes no signing queries using the signing with oracle reprogramming capabilities of the signature scheme.

**LEMMA B.4.** (Lemma 1 in [11]) Let BK be a key-blinding signature scheme which admits signing with oracle reprogramming with  $L1$  distance  $\delta_{\text{BK}}$  and min-entropy of reprogrammed points  $h_{\text{min, BK}}$ . Let  $\mathcal{A}$  be an adversary making  $q_B$  queries to the signing oracle,  $q_S$  queries to the signing oracle, and  $q_H$  queries to the random oracle. Using  $\mathcal{A}$ , we construct a key-only adversary  $\mathcal{A}^{q_S=0}$  who makes no signing queries for which  $\text{Adv}_{\mathcal{A}}^{\text{unlinkable}} \leq \text{Adv}_{\mathcal{A}^{q_S=0}}^{\text{unlinkable}} + q_H q_S 2^{-h_{\text{min, BK}}} + q_S \delta_{\text{BK}}$ .

This lemma in no way depends on whether the signature scheme admits unblinding or not, and so we simply refer to the proof in [11]. Next we need to consider the advantage of an adversary who makes no signing queries. To do this, we need to consider the distribution (over the randomness in the random oracle) induced by blinding the public key. We consider the difference between the blinding of a sequence of different public keys and blindings of the same public key, for randomly generated  $bk$ .

*Definition B.5.* (Independent Blinding) Let BK be a key-blinding signature scheme and let  $n$  be a positive integer. Let  $pk_0, pk_1, \dots, pk_n$  be public keys generated from BK.KeyGen. Sample blinding keys  $bk_1, \dots, bk_n$  from BK.BIGen. The blinding advantage of an adversary  $\mathcal{A}$ , denoted  $\text{Adv}_{\mathcal{A}}^{\text{IndBlind}, n}$  is the advantage of  $\mathcal{A}$  in distinguishing the following two distributions:

- 1) BK.BIPubKey( $pk_0, bk_1$ ),  $\dots$ , BK.BIPubKey( $pk_0, bk_n$ )
- 2) BK.BIPubKey( $pk_1, bk_1$ ),  $\dots$ , BK.BIPubKey( $pk_n, bk_n$ ).

This formulation of independent blinding is actually slightly simpler than the one found in [11], which had to contend with the identity public key being passed in as part of the context in order to make the blinding process one-way. This also allows us to simplify the proof of the next Lemma.

**LEMMA B.6.** (Variation on Lemma 2 of [11]) Let BK be a key-blinding signature scheme. Let  $\mathcal{A}^{q_S=0}$  be an adversary in the unlinkability game (Figure 2) that makes no queries to its signing oracle. Then there exists an algorithm  $\mathcal{B}$  such that  $\text{Adv}_{\mathcal{A}^{q_S=0}}^{\text{unlinkable}} \leq \text{Adv}_{\mathcal{B}}^{\text{IndBlind}, n}$ , where  $n$  is the number of queries that  $\mathcal{A}^{q_S=0}$  makes to the BLPUBLICKEY oracle.

**PROOF.** We modify the BLPUBLICKEY oracle so that rather than blinding  $pk_S$  each time, a new public key is derived from BK.KeyGen and blinded. Since the BLKEYSIGN oracle goes unqueried, this means only the CHALLENGE oracle is dependent on the identity public key  $pk_S$ , and specifically  $pk_S$  is only used in the  $b = 1$  branch of execution. So, we can in fact delay the sampling of  $sk_S, pk_S$  to exactly this point in the game (to just before line 7 of CHALLENGE() in Figure 2). But then these two branches depending on the bit  $b$  are entirely identical, and so the adversary's advantage in determining  $b$  is zero.

By noting that all we have done is replace  $pk_S$  with random, new public keys just as in Definition B.5, we obtain the desired result.  $\square$

Combining Lemmas 1 and 2 we obtain the proof of unlinkability for when bidirectional key-blinding is permitted we obtain the following:

**THEOREM B.7.** *Let BK be a key-blinding scheme that admits signing with oracle reprogramming with L1 statistical distance on forgeries  $\delta_{\text{BK}}$  and min-entropy of programmed points  $h_{\text{min,BK}}$ . Let  $\mathcal{A}$  be an adversary who attacks the unlinkability game with advantage  $\text{Adv}_{\mathcal{A}}^{\text{unlinkable}}$ , making  $q_S$  signing queries,  $q_B$  blinding queries, and  $q_H$  random oracle queries. Then there exists an algorithm  $\mathcal{B}$  distinguishing the independent blinding distributions with advantage at least*

$$\text{Adv}_{\mathcal{B}}^{\text{IndBlind},q_B} \geq \text{Adv}_{\mathcal{A}}^{\text{unlinkable}} - q_H q_S 2^{-h_{\text{min,BK}}} - q_S \delta_{\text{BK}}.$$

## C HARD PROBLEMS ON HASH FUNCTIONS

### C.1 Classical Problems

We denote by  $\text{Adv}_{\text{collision}}[\mathcal{A}, H]$ ,  $\text{Adv}_{\text{rpr}}[\mathcal{A}, H]$ , and  $\text{Adv}_{\text{zpr}}[\mathcal{A}, H]$  the advantage of an adversary  $\mathcal{A}$  to break the collision resistance, random preimage resistance, and zero preimage resistance of a hash function  $H$  respectively; cf. [17, Sec. 4.2] for proper definitions.

### C.2 Problems on Pairs of Hash Functions

We introduce the following advantages, which we use in the proof of unforgeability for ECDSA, and prove that they are negligible in the random oracle model (superscripted by ‘ro’).

*Definition C.1 (Product Intractibilities).* Let  $H_1, H_2$  be hash functions whose output spaces are  $\mathbb{Z}_q$  and  $\mathbb{Z}_q^*$ . Let  $\mathcal{A}$  be an adversary.

We define  $\text{Adv}_{1\text{prod}}[\mathcal{A}, H_1, H_2]$  as the advantage of  $\mathcal{A}$  breaking the “1prod” property of  $H_1$  and  $H_2$ , defined as the probability that  $\mathcal{A}$  is able to find a tuple  $(m, m', \beta)$  such that

$$H_1(m) = H_1(m')H_2(\beta)$$

We define  $\text{Adv}_{2\text{prod}}[\mathcal{A}, H_1, H_2]$  as the advantage of  $\mathcal{A}$  breaking the “2prod” property of  $H_1$  and  $H_2$ , defined as the probability that  $\mathcal{A}$  is able to find two pairs  $(m, \beta), (m', \beta')$  such that

$$H_1(m)H_2(\beta') = H_1(m')H_2(\beta).$$

**LEMMA C.2.** *If  $H_1$  and  $H_2$  are modeled as random oracle, then  $\text{Adv}_{1\text{prod}}^{\text{ro}}[\mathcal{A}, H_1, H_2] \leq Q_1^2 Q_2 / q$  and  $\text{Adv}_{2\text{prod}}^{\text{ro}}[\mathcal{A}, H_1, H_2] \leq Q_1^2 Q_2^2 / q$ , where  $Q_i$  for  $i \in \{1, 2\}$  is a bound on the number of queries to  $H_i$  made by  $\mathcal{A}$ .*

**PROOF.** The probability that the  $i_0$ -th query to  $H_1$ ,  $i_1$ -th query to  $H_1$ ,  $j_0$ -th query to  $H_2$ ,  $j_1$ -th query to  $H_2$  are such that  $h_{i_0} = h_{i_1} h_{j_1} / h_{j_0}$  is  $1/q$  (and similarly for  $h_{i_0} = h_{i_1} h_{j_1}$ ). The results follow by inductive application of the union bound.  $\square$

*Definition C.3 (Quotient Intractibility).* Let  $H_1, H_2$  be hash functions whose output spaces are  $\mathbb{Z}_q$  and  $\mathbb{Z}_q^*$ . Let  $\mathcal{A}$  be an adversary. We define  $\text{Adv}_{\text{quot}}[\mathcal{A}, H_1, H_2]$  as the the advantage of  $\mathcal{A}$  breaking the quotient property of  $H_1$  and  $H_2$ , defined as the probability that  $\mathcal{A}$  wins the following game.

- $\mathcal{A}$  chooses  $e \in \mathbb{Z}_q$  and sends it to the challenger;
- The challenger chooses  $t \in \mathbb{Z}_q^*$  uniformly at random and gives  $t$  to  $\mathcal{A}$ ;
- $\mathcal{A}$  outputs  $(m', \beta')$  and wins if  $te = H_1(m')/H_2(\beta')$ .

**LEMMA C.4.** *If  $H_1$  and  $H_2$  are modeled as random oracle, then  $\text{Adv}_{\text{quot}}^{\text{ro}}[\mathcal{A}, H_1, H_2] \leq Q_1 Q_2 / q$  where  $Q_i$  is a bound on the number of queries to  $H_i$  made by  $\mathcal{A}$ .*

**PROOF.** If  $e = 0$ , the probability that  $\mathcal{A}$  succeeds is bounded by  $Q_1/q$ . Otherwise, the probability that the  $i$ -th query to  $H_1$  and the  $j$ -th query to  $H_2$  are such that  $h_1 \cdot te = h_2$  is  $1/q$ . The result follows by inductive application of the union bound.  $\square$