# Compact Cut-and-Choose: Boosting the Security of Blind Signature Schemes, Compactly

Rutchathon Chairattana-Apirom
Brown University
rutchathon.c@gmail.com

Anna Lysyanskaya
Brown University
anna@cs.brown.edu

December 31, 2021

**Abstract**

Blind signature schemes are one of the best and best-studied tools for privacy-preserving authentication. It has a blind signing protocol in which a signer learns nothing about the message being signed or the resulting signature; thus such a signature can serve as an anonymous authentication token. Thus, constructing efficient blind signatures secure under realistic cryptographic assumptions is an important goal.

A recent paper by Benhamouda, Lepoint, Loss, Orrù, and Raykova (Eurocrypt '21) showed that a large class of blind signature schemes secure in the stand-alone setting are no longer secure when multiple instances of the blind signing protocol are executed concurrently. The best known technique to salvage the security of such blind signatures was recently proposed by Katz, Loss, and Rosenberg (Asiacrypt '21). For the security parameter $\kappa$, their technique transforms blind signature schemes that are secure for $\mathcal{O}(\log \kappa)$ concurrent executions of the blind signing protocol into ones that are secure for any $N = \mathsf{poly}(\kappa)$ concurrent executions. The resulting, transformed blind signing protocol needs $\mathcal{O}(N)$ times more computation and communication than the original one.

In this paper, we give an improved transform for obtaining a secure blind signing protocol tolerating $N = \mathsf{poly}(\kappa)$ concurrent executions from one that is secure for $\mathcal{O}(\log \kappa)$ concurrent executions. Our technique still needs $\mathcal{O}(N)$ times more computation, but only $\mathcal{O}(\log N)$ more communication than the original blind signature.

## 1 Introduction

A blind signing protocol is a protocol between a signer and a user in which the user outputs a digital signature on the desired message, while the signer learns nothing about the message or the resulting signature. A blind signature scheme is a signature scheme that has a blind signing protocol. In spite of having a relatively long history (they were introduced almost forty years ago by David Chaum [7]), blind signatures are a subject of excitement in the cryptography research community at the moment because they can be used as privacy-preserving authentication tokens that can replace browser cookies in certain applications, for example by the VPN by Google One (https://one.google.com/about/vpn/howitworks).

The formal definition of security of blind signatures [18, 13, 1, 21] requires two security properties: *blindness* and *one-more unforgeability*. Blindness guarantees that an adversarial signer can neither learn the message in the signing protocol nor link a particular message-signature pair to

a protocol execution. One-more unforgeability guarantees that an adversary cannot produce more signed messages than the number of times it invoked the signing protocol.

It is important that security hold even as the the blind signing protocol is executed together with other protocols. At a minimum, therefore, the blind signing protocol needs to be concurrently self-composable. Unfortunately, when executed concurrently, some otherwise attractive blind signing protocols [15, 20, 2, 6, 16, 4, 22, 23, 10] are no longer one-more-unforgeable; not in the sense that their proofs of security no longer apply, but recently a concrete and practical attack was discovered [5].

The starting point for our work is a recent paper by Katz, Loss and Rosenberg (KLR) [14] that showed how to boost the concurrent security of certain blind signing protocols, from being able to tolerate $\mathcal{O}(\log \kappa)$ concurrent executions to being able to tolerate $N = \mathsf{poly}(\kappa)$ executions, where $\kappa$ is the security parameter. This is best explained by using the Okamoto-Schnorr blind signature [15] as an example; Hauck, Kiltz and Loss [12] gave a general formulation of schemes that follow the same structure and to which the KLR technique applies.

Let $G$ be a group of prime order in which the discrete logarithm problem is hard, and let $g$ and $h$ be its generators. Suppose that we have a prover and an honest verifier who are given as input a group element $W$; the prover is additionally given $s$ and $t$ such that $W = g^s h^t$. Recall that there is an efficient Sigma-protocol (i.e. a three-move honest-verifier zero-knowledge proof of knowledge system with some additional properties) for proving knowledge of $s$ and $t$.

The Okamoto-Schnorr digital signature scheme can be seen as converting this proof system into a digital signature scheme using the Fiat-Shamir heuristic. More precisely, the public key is $W = g^s h^t$ and the secret key is the pair $(s, t)$. A Sigma-protocol for proving knowledge of $(s, t)$ starts with the message from the Prover, followed by a challenge from the Honest Verifier, followed by the response from the Prover; this protocol can be converted to a signing algorithm if the message from the Verifier is replaced by the output of a hash function modeled as a random oracle for the purposes of proving security. In the Okamoto-Schnorr blind signing protocol, the user executes the Verifier's side of this Sigma protocol, while the signer executes the Prover's side. The user, therefore, queries the hash function himself, never showing the message to the signer; moreover, the user blinds the values received from and sent to the signer, so that they cannot be linked to the resulting signature.

Pointcheval and Stern [18] developed the machinery for proving blind signatures such as Okamoto-Schnorr secure in the random-oracle model; however, their proof approach cannot tolerate more than $\mathcal{O}(\log \kappa)$ concurrent executions of this signature scheme; Benhamouda et al. [5] showed that this is not an accident, since an adversary orchestrating $\kappa$ concurrent executions can break one-more-unforgeability. Therefore, a different approach is needed to obtain a concurrently secure blind signature from Okamoto-Schnorr and other blind signatures with similar structure; Hauck, Kiltz and Loss refer to such blind signatures as *blind signatures from linear function families* and also show that they are secure under $\mathcal{O}(\log \kappa)$ concurrent executions [12].

The KLR technique requires that the signer keep track of the number $N$ of concurrent executions. In order to get one signature, the user creates $N$ instances of the Okamoto-Schnorr user; each instance uses its own random coins; the user commits to the randomness of each of these instances. The first message from the signer to the user consists of $N$ Okamoto-Schnorr first messages; the response from the user to the signer consists of the $N$ responses generated by these $N$ instances. Next, the signer asks to see the random coins of all but one instance (chosen uniformly at random), and aborts if the user cannot provide the correct random coins that explain all the messages ex-

changed so far. If the user passes this check, then the signer completes the signing protocol for the one instance whose randomness he didn't obtain. KLR's cut-and-choose blind signature, in other words, chooses one out of $N$ instances to complete, but only if the other $N-1$ are executed correctly. Even though the randomness for all but one instance is revealed, the message that is being signed stays hidden because in instance $i$, $\mu_i = \mathcal{H}'(m, r_i)$ is signed instead, where $\mathcal{H}'$ is another hash function, and random $r_i$ is *not* revealed to the signer as part of the check.

As we can see, the drawback of the KLR technique is that it increases both the communication and the computational complexity of the resulting protocol by a factor of $N$. In this paper, we show how to batch the messages between the signer and the user such that the communication complexity increases only by a factor of $\log N$.

Our first observation is that the random coins used in each instance can be derived from the same master seed using a pseudorandom function; in order to reveal the random coins of all but one of the instances it is sufficient to produce just $\log N$ values using techniques similar to those in the pseudorandom function construction of Goldreich, Goldwasser and Micali [11] and puncturable PRFs of Sahai and Waters [19]. Thus, the random coins that the user sends to the signer need not take up more than $\mathcal{O}(\log N)$ extra multiplicative factor. Our second insight is that in order to batch the $\mu_i$'s, we use randomizable commitments: the user commits to $m$ and sends the resulting commitment $C$ to the signer. In each of the $N$ instances, the user will randomize this commitment to obtain the instance-specific commitment $\mu_i$; the randomness to do this step is derived from the master seed and therefore available to the signer during the verification step.

But how do we batch the messages from the signer to the user? Okamoto-Schnorr signatures and their generalization to blind signatures from linear function families have homomorphic properties that allow batching the messages from the signer. We present the details of the construction in Section 3; in Section 5 we show that when batched this way, the blind signature is secure under $N = \mathsf{poly}(\kappa)$ concurrent executions.

## 2 Preliminaries

NOTATION. For any positive integer $n$, let $[n] = \{1, 2, \ldots, n\}$. The notation $a \leftarrow S$ where $S$ is a set means that $a$ is a value uniformly randomly sampled from the set $S$. While for an algorithm $\mathcal{A}$, $a \leftarrow \mathcal{A}(x)$ denotes $a$ as an output of the algorithm $\mathcal{A}$ with input $x$. Denote $\kappa$ as the security parameter. A function $\epsilon : \mathbb{N} \to [0, 1]$ is negligible if for some $c > 0$, for all polynomial $p$ and $\kappa \geq c$, $\epsilon(\kappa) < 1/p(\kappa)$.

### 2.1 Blind Signatures

**Definition 2.1** (Blind Signature Schemes [21])**.** *A blind signature scheme consists of four algorithms* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{User}, \mathsf{Verify})$ *defined as follows:*

- ***Key generation:*** $\mathsf{KeyGen}(1^\kappa)$ *takes as input the security parameter* $1^\kappa$ *and generates the pubic-secret key pair* $(\mathsf{pk}, \mathsf{sk})$ *along with the signer's internal states* $\mathsf{state}$.
- ***Signing algorithm:*** $\mathsf{Sign}(\mathsf{sk}, \mathsf{state})$ *is an interactive algorithm taking as input the secret key* $\mathsf{sk}$ *and the state variable* $\mathsf{state}$. *It interacts with the user algorithm* $\mathsf{User}$ *and outputs either* $\perp$*(aborting) or* $1$*(completion) and the updated state* $\mathsf{state}'$.
- ***User algorithm:*** $\mathsf{User}(\mathsf{pk}, m)$ *is an interactive algorithm taking as input its message* $m$ *and the public key* $\mathsf{pk}$. *It interacts with the signing algorithm* $\mathsf{Sign}$ *and outputs either* $\perp$*(aborting) or*

*the signature $\sigma$.*

***Verification:*** $\mathsf{Verify}(\mathsf{pk}, m, \sigma)$ *takes as input the public key* $\mathsf{pk}$*, a message* $m$*, and a signature* $\sigma$*. It outputs either "accept" or "reject".*

*We say that a blind signature scheme is correct if for all messages* $m$*, secret-public key pairs* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\kappa)$*, and states* $\mathsf{state}$ *returned from* $\mathsf{KeyGen}$ *or* $\mathsf{Sign}$*, if* $\mathsf{Sign}(\mathsf{sk}, \mathsf{state})$ *and* $\mathsf{User}(\mathsf{pk}, m)$ *are executed honestly with* $\sigma$ *as the user's output, then* $\mathsf{Verify}(\mathsf{pk}, m, \sigma)$ *always accepts.*

**Definition 2.2** (Blindness [21])**.** *A blind signature scheme* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{User}, \mathsf{Verify})$ *is blind if for any probabilistic polynomial-time adversary* $\mathcal{A}$*,* $\mathcal{A}$ *succeeds in the following game with negligible advantage over* $\frac{1}{2}$*:*

- $(m_0, m_1, \mathsf{pk}, \mathsf{state}_\mathcal{A}) \leftarrow \mathcal{A}(1^\kappa)$*.* $b \leftarrow \{0, 1\}$*.*
- *Run* $\mathcal{A}^{\mathsf{User}(\mathsf{pk}, m_b), \mathsf{User}(\mathsf{pk}, m_{1-b})}(1^\kappa, \mathsf{state}_\mathcal{A})$ *where* $\mathcal{A}$ *can interleave the execution between the two oracles.*
- *Let* $\sigma_b, \sigma_{1-b}$ *be the outputs of the respective oracles. If* $\sigma_0 = \perp$ *or* $\sigma_1 = \perp$*, send* $\perp$ *to* $\mathcal{A}$*. If not, send* $(\sigma_0, \sigma_1)$ *to* $\mathcal{A}$*.*
- $\mathcal{A}$ *outputs* $b'$*. It succeeds if* $b' = b$*.*

**Definition 2.3** (One-more Unforgeability [21])**.** *Let* $\ell : \mathbb{N} \to \mathbb{N}$*. A blind signature scheme* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{User}, \mathsf{Verify})$ *is* $\ell$*-one-more unforgeable if for any probabilistic polynomial-time adversary* $\mathcal{A}$*,* $\mathcal{A}$ *succeeds in the following game with negligible probability.*

- $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\kappa)$*.*
- *Run* $\mathcal{A}^{\mathsf{Sign}(\mathsf{sk})}(\mathsf{pk})$*, where* $\mathcal{A}$ *may initiate an arbitrary number of oracle executions, but* $\mathsf{Sign}(\mathsf{sk})$ *will only complete at most* $\ell = \ell(\kappa)$ *executions.*
- $\mathcal{A}$ *outputs* $\ell + 1$ *message-signature pairs* $(m_1, \sigma_1), \ldots, (m_{\ell+1}, \sigma_{\ell+1})$*.*
- $\mathcal{A}$ *wins if all pairs are distinct and* $\mathsf{Verify}(\mathsf{pk}, m_i, \sigma_i)$ *accepts for all* $i \in [\ell + 1]$*.*

## 2.2 Randomizable Commitment

In this subsection, we define a randomizable commitment scheme. In our construction, such a scheme will be used by the signature recipient, who will commit to the message to be be signed. Our definition is based on that of Ananth, Deshpande, Kalai and Lysyanskaya [3] with slight differences: (1) their definition is for bit commitment schemes and (2) the randomization factor in our definition can come from a different domain.

**Definition 2.4** (Randomizable Commitment [3])**.** *A randomizable commitment consists of three algorithms* $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Randomize})$ *defined as follows:*

$\mathsf{Setup}(1^\kappa)$ *generates parameters* $\mathsf{params}$ *used for the commitment schemes defining the message space* $\mathcal{M}$*, randomness space* $\mathcal{X}$*, and randomization factor space* $\mathcal{X}'$*.*

$\mathsf{Commit}(\mathsf{params}, m; r)$ *commits a message* $m$ *with randomness* $r$*, returning a commitment* $\mathsf{com}$*.*

$\mathsf{Randomize}(\mathsf{params}, \mathsf{com}; r')$ *randomizes the commitment* $\mathsf{com}$ *with randomization factor* $r' \in \mathcal{X}'$*, returning a new commitment* $\mathsf{com}'$*. If* $\mathsf{com} \leftarrow \mathsf{Commit}(\mathsf{params}, m; r)$*, then* $\mathsf{com}'$ *can be revealed with* $(m, \Phi_{\mathsf{params}}(r, r'))$ *where* $\Phi$ *is an operation on* $r, r'$*.*

When clear from the context, we will omit $\mathsf{params}$ and use $\mathsf{Commit}(m; r), \mathsf{Randomize}(\mathsf{com}; r')$, and $\Phi$ for simplicity.

**Definition 2.5** (Correctness)**.** *A randomizable commitment scheme* $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Randomize})$ *is correct if for any* $\mathsf{com}$ *which can be opened to* $(m, r)$ *and any* $r' \in \mathcal{X}$*,* $\mathsf{Randomize}(\mathsf{com}; r')$ *can be opened to* $(m, \Phi(r, r'))$*.*

Our construction requires a randomizable commitment scheme with hiding and randomizability as defined below:

**Definition 2.6** (Hiding)**.** *A randomizable commitment scheme* $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Randomize})$ *is (statistically/computationally) hiding if for all* $m_0, m_1$*, the distributions* $\{\mathsf{Commit}(m_0; r) : r \leftarrow \mathcal{X}\}$ *and* $\{\mathsf{Commit}(m_1; r) : r \leftarrow \mathcal{X}\}$ *are (statistically/computationally) indistinguishable i.e. for any (unbounded/probabilistic polynomial time) adversary* $\mathcal{A}$*,* $\mathcal{A}$ *cannot guess whether a given distribution is based on* $m_0$ *or* $m_1$*, which are messages chosen by* $\mathcal{A}$*, with non-negligible advantage over* $\frac{1}{2}$*.*

*Specifically, we say that a randomizable commitment is* $(t, \epsilon_{\mathsf{Hide}})$*-hiding if for any adversary* $\mathcal{A}$ *running in time* $t$*,* $\mathcal{A}$ *cannot guess whether a given distribution is based on commitments to* $m_0$ *or* $m_1$*, which are messages chosen by* $\mathcal{A}$*, with more than* $\epsilon_{\mathsf{Hide}}$ *advantage over* $\frac{1}{2}$*.*

**Definition 2.7** (Randomizable)**.** *A randomizable commitment scheme* $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Randomize})$ *is* $(t, \epsilon_{\mathsf{Rand}})$*-randomizable if for any adversary* $\mathcal{A}$ *running in time* $t$*,* $\mathcal{A}$*'s advantage over* $\frac{1}{2}$ *in this game is at most* $\epsilon_{\mathsf{Rand}}$*:*

- $\mathsf{params} \leftarrow \mathsf{Setup}(1^\kappa)$.
- $(m, \mathsf{state}) \leftarrow \mathcal{A}(1^\kappa, \mathsf{params})$
- $\mathsf{com} \leftarrow \mathsf{Commit}(m; r)$ *where* $r \leftarrow \mathcal{X}$
- $b \leftarrow \{0, 1\}, r' \leftarrow \mathcal{X}'$
- *If* $b = 0$*,* $open = r'' \leftarrow \mathcal{X}$*. Else,* $open = \Phi(r, r')$
- $b' \leftarrow \mathcal{A}(\mathsf{com}, open, \mathsf{state})$
- $\mathcal{A}$ *succeeds if* $b' = b$

(Our definition of randomizability is a relaxed version of the one given by Ananth, Deshpande, Kalai and Lysyanskaya [3], which requires that for every randomness $r$, if $r'$ is chosen uniformly at random, then $\Phi(r, r')$ is uniformly random; the relaxation covers the Fujisaki-Okamoto commitment [9, 8].)

**Definition 2.8** (Binding)**.** *A randomizable commitment* $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Randomize})$ *is* $(t, \epsilon_{\mathsf{Bind}})$*-binding if for any adversary* $\mathcal{A}$ *running in time* $t$*,* $\mathcal{A}$*'s advantage in the following game is at most* $\epsilon_{\mathsf{Bind}}$*:*

- $\mathsf{params} \leftarrow \mathsf{Setup}(1^\kappa)$.
- $(m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}(1^\kappa, \mathsf{params})$
- $\mathcal{A}$ *succeeds if* $\mathsf{Commit}(m_0; r_0) = \mathsf{Commit}(m_1; r_1)$.

### 2.2.1 Examples of Randomizable Commitment Schemes

An example of a randomizable commitment is the Pedersen commitment [17] defined as follows:

**Definition 2.9** (Pedersen Commitment)**.** *The Pedersen Commitment consists of three algorithms* $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Randomize})$*:*

$\mathsf{Setup}(1^\kappa)$ *generates a cyclic group* $\mathbb{G}$ *of prime order* $q$ *and two generators* $g, h \in \mathbb{G}$ *where the message space, randomness space, and randomization factor space are all* $\mathbb{Z}_q$*.*

$\mathsf{Commit}(m; r)$ *outputs* $g^m h^r$*.*

$\mathsf{Randomize}(\mathsf{com}; r')$ *outputs* $\mathsf{com}' = \mathsf{com} \cdot h^{r'}$*. If* $\mathsf{com} = \mathsf{Commit}(m; r)$*,* $\mathsf{com}'$*can be opened as* $(m, r + r')$*.*

The Pedersen commitment is perfectly hiding($(t,0)$-hiding for any $t$), perfectly randomizable($(t,0)$-randomizable for any $t$), and computationally binding($(t,\epsilon_{\mathsf{Bind}})$-binding where $\epsilon_{\mathsf{Bind}}$ is negligible and $t = \mathsf{poly}(\kappa)$).

Another example is the Fujisaki-Okamoto commitment scheme [9, 8]:

**Definition 2.10** (Fujisaki-Okamoto Commitment)**.** *The Fujisaki-Okamoto Commitment consists of three algorithms* $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Randomize})$*:*

$\mathsf{Setup}(1^\kappa)$ *generates an integer* $N = pq$ *such that* $p = 2p'+1, q = 2q'+1$ *and* $p, q, p', q'$ *are primes and picks* $g, h \leftarrow QR_N$*. The algorithm also defines the message space* $[0, N)$*, the randomness space* $[0, 2^{2|N|}N)$*, and the randomization factor space* $[0, 2^{|N|}N)$*.*

$\mathsf{Commit}(m; r)$ *outputs* $g^m h^r \pmod N$*.*

$\mathsf{Randomize}(\mathsf{com}; r')$ *outputs* $\mathsf{com}' = \mathsf{com} \cdot h^{r'} \pmod N$*. If* $\mathsf{com} = \mathsf{Commit}(m; r)$*,* $\mathsf{com}'$ *can be opened as* $(m, r + r')$*.*

The Fujisaki-Okamoto commitment scheme is statistically hiding (more precisely, $(t,\epsilon)$-hiding for any $t$, where $\epsilon$ is negligible), $(t, \epsilon_{\mathsf{Rand}})$-randomizable for any $t$ and with $\epsilon_{\mathsf{Rand}}$ negligible, and computationally binding assuming that factoring such an integer $N$ is hard.

## 2.3 Linear Function Family

**Definition 2.11** ([12])**.** *A* ***linear function family*** $\mathsf{LF}$ *is a tuple of algorithms* $(\mathsf{PGen}, \mathsf{F}, \Psi)$ *defined as follows:*

$\mathsf{PGen}(1^\kappa)$ *returns system parameters* $\mathsf{params}$ *which define abelian groups* $\mathcal{S}, \mathcal{D}, \mathcal{R}$ *with* $|\mathcal{S}|, |\mathcal{R}| \geq 2^\kappa$ *and there exists scalar multiplication* $\cdot \ : \ \mathcal{S} \times \mathcal{D} \to \mathcal{D}$ *with* $s \cdot (x + x') = s \cdot x + s \cdot x'$ *for all* $s \in \mathcal{S}$ *and* $x, x' \in \mathcal{D}$*. The same applies for* $\mathcal{R}$*. Note that it is not necessarily true that* $(s + s') \cdot x = s \cdot x + s' \cdot x$*. Hauck et al. call this a pseudo-module.*

$\mathsf{F}_{\mathsf{params}}(x)$ *takes as input the system parameters and an element from* $\mathcal{D}$*.* $\mathsf{F}_{\mathsf{params}}(x)$ *returns an element in* $\mathcal{R}$*. As an abuse of notation, we will use* $\mathsf{F}(\cdot)$ *as* $F_{\mathsf{params}}(\cdot)$*. We require that:*

*(1)* $\mathsf{F}$ *is a pseudo-module homomorphism: for all* $s \in \mathcal{S}, x, y \in \mathcal{D}$*,* $\mathsf{F}(s \cdot x + y) = s \cdot \mathsf{F}(x) + \mathsf{F}(y)$

*(2)* $\mathsf{F}$ *has a pseudo torsion-free element in the kernel: there exists* $z^* \in \mathcal{D}$ *such that* $\mathsf{F}(z^*) = 0$ *and for all distinct* $s, s' \in \mathcal{S}, s \cdot z^* \neq s' \cdot z^*$*.*

*(3)* $\mathsf{F}$ *is smooth: if* $x \leftarrow \mathcal{D}$ *is sampled uniformly,* $\mathsf{F}(x)$ *has uniform distribution in* $\mathcal{R}$*.*

$\Psi_{\mathsf{params}}(y, s, s')$ *takes as inputs* $y \in \mathcal{R}$*, and* $s, s' \in \mathcal{S}$*, and returns a value in* $\mathcal{D}$*. The function satisfies for all* $y$ *in the range of* $\mathsf{F}$ *and* $s, s' \in \mathcal{S}$*,*

$$(s + s') \cdot y = s \cdot y + s' \cdot y + \mathsf{F}(\Psi_{\mathsf{params}}(y, s, s'))$$

*Intuitively, the distributor function* $\Psi$ *outputs a correction term that corrects for the fact that the group operation in* $\mathcal{S}$ *may not distribute over* $\mathcal{R}$*.*

For example (from [14]), the Schnorr blind signature has an underlying linear function family $\mathsf{LF} = (\mathsf{PGen}, \mathsf{F}, \Psi)$ such that $\mathsf{PGen}$ generates system parameters defining $\mathcal{S} = \mathcal{D} = \mathbb{Z}_q$ and $\mathcal{R} = \mathbb{G}$ as a cyclic group of order $q$ with generator $g$. $\mathsf{F}$ maps $x$ to $g^x$. However, $\mathsf{F}$ in this setting does not have a pseudo torsion-free element, because $\mathsf{F}$ is a bijection. $\Psi$ is a zero function.

We define two security properties for a linear function family:

**Definition 2.12** (Preimage resistant [14]). *A linear function family* $\mathsf{LF}$ *is* ***preimage resistant*** *if for any adversary* $\mathcal{A}$*,* $\mathcal{A}$*'s success probability in the following game is negligible:*

- $\mathsf{params} \leftarrow \mathsf{PGen}(1^\kappa)$
- $x \leftarrow \mathcal{D}$
- $x' \leftarrow \mathcal{A}(\mathsf{params}, \mathsf{F}(x))$
- $\mathcal{A}$ *succeeds if* $\mathsf{F}(x) = \mathsf{F}(x')$

$\mathsf{LF}$ *is* $(t, \epsilon_{PRE})$*-preimage resistant if for every* $\mathcal{A}$ *with running time at most* $t$*,* $\mathcal{A}$ *succeeds with probability at most* $\epsilon_{PRE}$*.*

**Definition 2.13** (Collision resistant [14]). *A linear function family* $\mathsf{LF}$ *is* ***collision resistant*** *if for any adversary* $\mathcal{A}$*,* $\mathcal{A}$*'s success probability in the following game is negligible:*

- $\mathsf{params} \leftarrow \mathsf{PGen}(1^\kappa)$
- $(x, x') \leftarrow \mathcal{A}(\mathsf{params})$
- $\mathcal{A}$ *succeeds if* $\mathsf{F}(x) = \mathsf{F}(x')$ *and* $x \neq x'$

$\mathsf{LF}$ *is* $(t, \epsilon_{CR})$*-collision resistant if for every* $\mathcal{A}$ *with running time at most* $t$*,* $\mathcal{A}$ *succeeds with probability at most* $\epsilon_{CR}$*.*

In the case of the Schnorr blind signature, it is preimage-resistant based on the discrete logarithm assumption and collision-resistant by its bijectivity.

## 2.4 Blind Signatures from a Linear Function Family

Hauck, Kiltz, Loss [12] defined the generic construction of a three-move blind signature schemes from any linear function family $\mathsf{LF}$ and a hash function $\mathcal{H}$ modeled as a random-oracle called $\mathsf{BS}[\mathsf{LF}]$. The construction of $\mathsf{BS}[\mathsf{LF}]$ is briefly outlined in Figure 1 and formally defined in Appendix A. It may be helpful to think about it as a generalization of the Okamoto-Schnorr blind signature scheme described in the Introduction, or of the Schnorr blind signature scheme. Hauck, Kiltz, Loss [12] proved the following theorem:

**Theorem 2.14** ([12]). *Let* $\mathsf{LF}$ *be a linear function family and* $\mathcal{H} : \{0,1\}^* \to \mathcal{S}$ *be a hash function modeled as a random oracle. If* $\mathsf{LF}$ *is* $(\epsilon', t')$*-collision resistant and has a pseudo torsion-free element in the kernel, then there exists no adversary* $\mathcal{A}$ *with success probability greater than* $\epsilon$ *in* $\ell$*-one-more unforgeability game against* $\mathsf{BS}[\mathsf{LF}]$ *that runs in time* $t$*, initiates at most* $p$ *protocol execution, and makes at most* $q_{\mathcal{H}}$ *queries to* $\mathcal{H}$ *where*

$$t' = 2t, \epsilon' = \mathcal{O}\left(\left(\frac{\epsilon}{2} - \frac{(Q \cdot (p-\ell))^{\ell+1}}{2^{2\kappa}}\right)^3 \frac{1}{Q^2 \ell^3}\right)$$

*and* $Q = q_{\mathcal{H}} + \ell + 1$*. Moreover,* $\mathsf{BS}[\mathsf{LF}]$ *is perfectly blind in the random oracle model.*

This theorem shows that the Okamoto-Schnorr blind signature is $\ell$-one-more unforgeable for $\ell = \mathcal{O}(\log \kappa)$. However, in the setting of the blind Schnorr signature, the linear function $\mathsf{F}$ does not contain pseudo torsion-free element in the kernel, so this result does not apply.

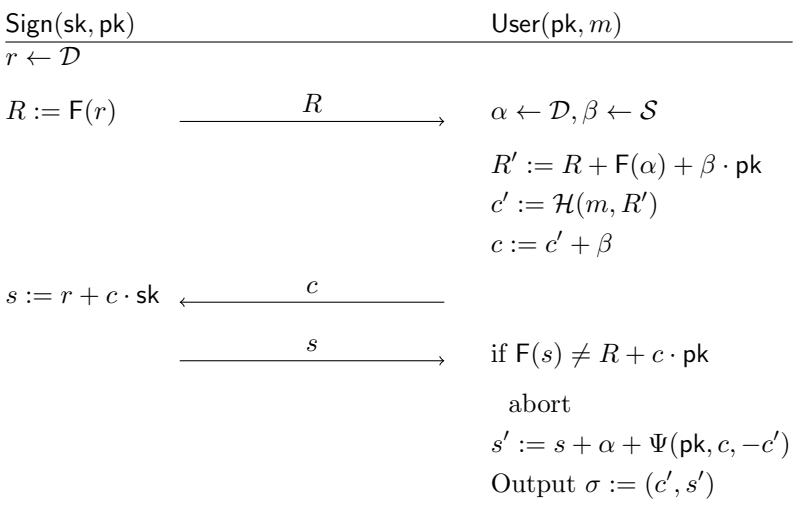


Figure 1: Interaction between the signer and the user in BS[LF] protocol. (Figure 1 in [14])

### 2.5 Cut-and-Choose Blind Signature Construction

Katz, Loss, and Rosenberg [14] introduced the cut-and-choose blind signature scheme CCBS[LF] from a linear function family LF. We briefly outline the CCBS[LF] construction in Figure 2 and formally define it in Appendix A. The construction is proven blind and $\ell$-one-more unforgeable for any $\ell = \mathsf{poly}(\kappa)$ if the underlying generic blind signature scheme from LF, BS[LF], is blind and $\lambda$-one-more unforgeable for $\lambda = \mathcal{O}(\log \kappa)$.

**Theorem 2.15** ([14]). *Let* LF *be a linear function family that is preimage resistant and* $\mathcal{H}, \mathcal{H}'$ *be random oracles. If* BS[LF] *satisfies blindness, then* CCBS[LF] *satisfies blindness. Moreover, if* BS[LF] *satisfies* $\lambda$*-one-more unforgeablility for* $\lambda = \mathcal{O}(\log \kappa)$*, then* CCBS[LF] *satisfies* $\ell$*-one-more unforgeability for* $\ell = \mathsf{poly}(\kappa)$.

According to the theorem, this transformation can be applied to the Okamoto-Schnorr blind signature to improve $\ell$-one-more unforgeability guarantee from $\ell = \mathcal{O}(\log \kappa)$ to $\ell = \mathsf{poly}(\kappa)$.

## 3 Our Compact Cut-and-Choose Technique

In this section, we define our Compact Cut-and-Choose blind signature scheme for a linear function family LF, abbreviated as CCCBS[LF]. Let $\mathcal{H}_1 : \{0,1\}^* \to \{0,1\}^{2\kappa}, \mathcal{H}_2 : \{0,1\}^* \to \{0,1\}^{\kappa}, \mathcal{H}_3 : \{0,1\}^* \to \mathcal{S} \times \mathcal{S} \times \mathcal{X}', \mathcal{H}_4 : \{0,1\}^* \to \{0,1\}^{\kappa}$ be random oracles. Also, let LF be a linear function family and (Setup, Commit, Randomize) be a randomizable commitment scheme. Then, the CCCBS[LF] construction is defined as follows:

**Initialization and key generation:** Generate the parameters as follows: Sample $\mathsf{params}_{\mathsf{Commit}} \leftarrow \mathsf{Setup}(1^\kappa)$ and $\mathsf{params}' \leftarrow \mathsf{LF.PGen}(1^\kappa)$; the latter defines the distributions $\mathcal{D}, \mathcal{R}, \mathcal{S}$. Set $\mathsf{params} = (\mathsf{params}', \mathsf{params}_{\mathsf{Commit}})$. Initialize the variables $N$ and ctr that are part of the signer's state as follows: $N := 2 = 2^2 - 2, \mathsf{ctr} := 0$. Sample $x \leftarrow \mathcal{D}$, and let $\mathsf{sk} = x$, $\mathsf{pk} = (\mathsf{params}, \mathsf{F}(x))$.

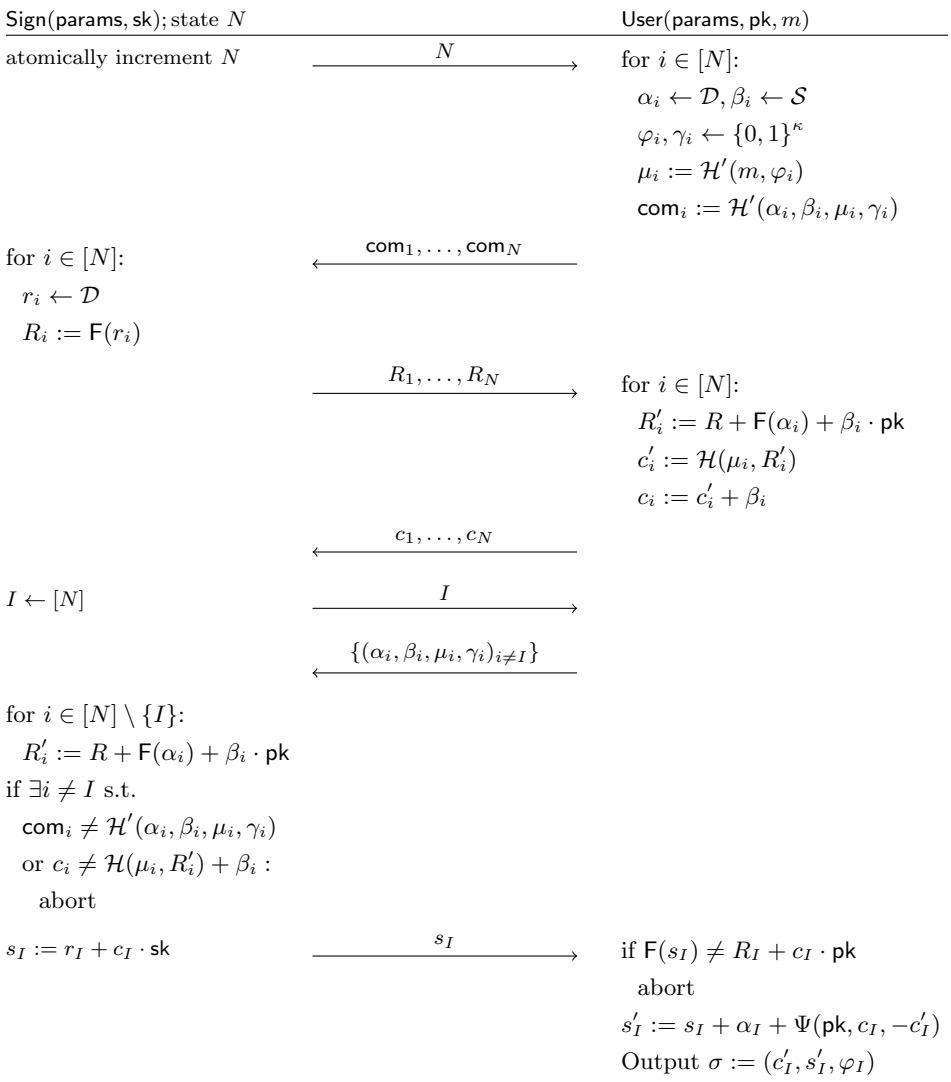


Figure 2: Interaction between the signer and the user in CCBS[LF] protocol. (Figure 2 in [14])

**Signing protocol:** the signer $\mathsf{Sign}(\mathsf{params}, \mathsf{sk})$ with state $(N, \mathsf{ctr})$ and the user $\mathsf{User}(\mathsf{pk}, m)$ are defined as follows; each item corresponds a round of communication in Figure 3:

1. First, the user commits to their message using the randomizable commitment scheme $C := \mathsf{Commit}(m; r)$ where $r$ is uniformly sampled from the domain of the randomness $\mathcal{X}$ for the randomizable commitment, and sends $C$ to the signer.
2. The signer atomically increments the counter $\mathsf{ctr}$ and, if $\mathsf{ctr} = N$, sets $N := 2N+2, \mathsf{ctr} := 0$. The signer sends $N$ to the user. Here, $N$ is always in the form of $2^l - 2$.
3. The user computes $N$ pseudorandom seeds for use in $N$ signing sessions (only one of which will ultimately be completed), as follows. First, it generates a random master seed $seed_{0,0} \leftarrow \{0,1\}^\kappa$; all seeds will be computed deterministically from the master seed in the same way as the GGM PRF construction [11] using $\mathcal{H}_1$ as the pseudorandom

generator in that construction. More precisely:

The seeds will be organized into a binary tree of depth $l = \log(N+2)$; the master seed $seed_{0,0}$ is stored at the root of the tree. By $seed_{i,j}$, we denote the seed that is stored at depth $i$, in position $j \geq 0$ from the left.

Recall that the random oracle $\mathcal{H}_1$ always outputs strings of length $2\kappa$; by $\mathcal{H}_{1,0}(x)$, let us denote the first $\kappa$ bits of $\mathcal{H}_1(x)$, while $\mathcal{H}_{1,1}(x)$ will denote the remaining bit (i.e. the second half of the bits) of $\mathcal{H}_1(x)$. For $0 < i \leq l$, $0 \leq j < 2^i$, $seed_{i,j} = \mathcal{H}_{1,j \bmod 2}(seed_{i-1,\lfloor j/2 \rfloor})$. At the end of this process, the user will obtain $s_{l,0}, \ldots, s_{l,N+1}$. Next, the user computes $h_i := \mathcal{H}_2(seed_{l,i})$ and sends $H_1 := \mathcal{H}_2(h_1, \ldots, h_N)$ to the signer.

4. The signer uniformly randomly generates $r_i \leftarrow \mathcal{D}, R_i := \mathsf{F}(r_i)$ for $i \in [l]$ and sends $R_1, \ldots, R_l$ to the user.
5. Upon receiving $R_i$'s, the user does the following for each $1 \leq i \leq N$:
   - Calculates $\tilde{R}_i := \sum_{j \in S_i} R_j$ where $S_i = \{j \in [l] : j^{th}\text{-bit of } i \text{ is } 1\}$, and generate $\alpha_i, \beta_i, \varphi_i := \mathcal{H}_3(seed_{l,i})$.
   - Blinds $\tilde{R}_i$ and $C$ into $\tilde{R}'_i := \tilde{R}_i + \mathsf{F}(\alpha_i) + \beta_i \cdot \mathsf{pk}$ and $\mu_i := \mathsf{Randomize}(C, \varphi_i)$ respectively.
   - Computes $c'_i := \mathcal{H}(\mu_i, \tilde{R}'_i)$ and blinds it to $c_i := c'_i + \beta_i$.

   Then, using $\mathcal{H}_4$, the user computes and sends a hash $H_2 := \mathcal{H}_4(c_1, \ldots, c_N)$ to the signer.
6. The signer uniformly generates $I \leftarrow [N]$ forwarding it to the user.
7. The user reveals enough information for the signer to be able to compute $seed_{l,i}$ for all $i \neq I$, such that $seed_{l,I}$ remains hidden, as follows: treat $I$ as an $l$-bit string; for $1 \leq o \leq l$, reveal the seed $seed_{o,j_o}$, where $j_o$ is a string of $o$ bits that agrees with $I$ on the first $o-1$ bits but disagrees on bit $o$.
8. The signer calculates all $seed_{l,i}$ for $i \neq I$, as follows: let $o \geq 1$ be an integer such that, when $i$ and $I$ are viewed as $l$-bit strings, $i$ agrees with $I$ on the first $o-1$ bits but disagrees on bit $o$. Then $seed_{l,i}$ was computed as a function of $seed_{o,j_o}$ which the user revealed to the signer in the previous step.

   Next, the signer uses $seed_{l,i}$ to compute $\alpha_i, \beta_i, \varphi_i$ which are used for calculating $c_i$ for all $i \neq I$.

   Then, the signer verifies that $H_1 = \mathcal{H}_2(\mathcal{H}_2(seed_{l,0}), \ldots, h_I, \ldots, \mathcal{H}_2(seed_{l,N+1}))$ received from step (3) and checks that $H_2 = \mathcal{H}_4(c_1, \ldots, c_N)$. If one of the checks does not verify, the signer aborts. Lastly, the signer calculates $s_I := \sum_{j:j \in S_I} r_j + c_I \cdot \mathsf{sk}$ and sends it to the user.
9. The user computes $s'_I := s_I + \alpha_I + \Psi(\mathsf{pk}, c_I, -c'_I)$ and outputs $\sigma = (c'_I, s'_I, \Phi(r, \varphi_I))$.

**Verification algorithm:** $\mathsf{Verify}(\mathsf{pk}, m, \sigma)$ takes as input a message $m$ and a signature $\sigma = (c, s, \varphi)$. It runs and returns the output of the verification algorithm of $\mathsf{BS}[\mathsf{LF}]$ on inputs $(\mathsf{Commit}(m; \varphi), (c, s))$.

We listed the differences of our construction $\mathsf{CCCBS}[\mathsf{LF}]$ and the $\mathsf{CCBS}[\mathsf{LF}]$ construction in Appendix B. Below, we show that our construction is correct.

**Theorem 3.1.** *The* $\mathsf{CCCBS}[\mathsf{LF}]$ *construction is correct i.e. for all* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa)$ *and messages* $m$*, if* $\sigma$ *is the output of the signing protocol* $\langle \mathsf{Sign}, \mathsf{User} \rangle$ *on message* $m$ *and public-secret key pair* $(\mathsf{pk}, \mathsf{sk})$*, then* $\mathsf{Verify}(\mathsf{pk}, m, \sigma) = \mathsf{accept}$.

| $\mathsf{Sign}(\mathsf{params}, \mathsf{sk})$; states $N, \mathsf{ctr}$ | | $\mathsf{User}(\mathsf{params}, \mathsf{pk}, m)$ |
|---|---|---|
| Atomically do: (2) | $\xleftarrow{C}$ | $r \leftarrow \mathcal{X}, C := \mathsf{Commit}(m; r)$ (1) |
| $\mathsf{ctr} := \mathsf{ctr} + 1$ | | |
| if $\mathsf{ctr} = N$ : | | |
| $N := 2N + 2, \mathsf{ctr} := 0$ | | |
| $l := \log(N + 2)$ | | |
| | $\xrightarrow{N}$ | $seed_{0,0} \leftarrow \{0,1\}^\kappa$ (3) |
| | | for $i = 1, \ldots, l$ : |
| | | for $j = 0, 2, \ldots, 2^i - 2$ : |
| | | $(seed_{i,j}, seed_{i,j+1}) := \mathcal{H}_1(seed_{i-1,j/2})$ |
| | | for $i \in [N]$: |
| | | $h_i = \mathcal{H}_2(seed_{l,i})$ |
| | | $H_1 := \mathcal{H}_2(h_1, \ldots, h_N)$ |
| for $i \in [l]$: (4) | $\xleftarrow{H_1}$ | |
| $r_i \leftarrow \mathcal{D}, R_i := \mathsf{F}(r_i)$ | | |
| | $\xrightarrow{R_1, \ldots, R_l}$ | for $i \in [N]$: (5) |
| | | $\tilde{R}_i := \sum_{j \in S_i} R_j$ |
| | | $\alpha_i, \beta_i, \varphi_i := \mathcal{H}_3(seed_{l,i})$ |
| | | $\tilde{R}'_i := \tilde{R}_i + \mathsf{F}(\alpha_i) + \beta_i \cdot \mathsf{pk}$ |
| | | $\mu_i := \mathsf{Randomize}(C; \varphi_i)$ |
| | | $c'_i := \mathcal{H}(\mu_i, \tilde{R}'_i), c_i := c'_i + \beta_i$ |
| | | $H_2 := \mathcal{H}_4(c_1, \ldots, c_N)$ |
| | $\xleftarrow{H_2}$ | |
| $I \leftarrow [N]$ (6) | $\xrightarrow{I}$ | Treating $I$ as an $l$-bit string (7) |
| | | for $o \in [l]$ : |
| | | $j_o :=$ $o$-bit string such that |
| | | $j_o[1 \ldots o-1] = I[1 \ldots o-1]$ |
| | | but $j_o[o] \neq I[o]$ |
| | | $seed'_o := seed_{o,j_o}$ |
| Reconstruct $seed_{l,i}$'s from $seed'_o$'s (8) | $\xleftarrow{\{seed'_o\}_{o \in [l]}, h_I, c_I}$ | |
| compute $c_i$'s for all $i \neq I$. | | |
| Verify $seed_{l,i}$'s with $H_1$ | | |
| and $c_1, \ldots, c_n$ with $H_2$ | | |
| if not valid, the signer aborts. | | |
| $s_I := \sum_{j \in S_I} r_j + c_I \cdot \mathsf{sk}$ | | |
| | $\xrightarrow{s_I}$ | $s'_I := s_I + \alpha_I + \Psi(\mathsf{pk}, c_I, -c'_I)$ (9) |
| | | Output $\sigma = (c'_I, s'_I, \Phi(r, \varphi_I))$ |

Figure 3: Signing protocol of our construction in Section 3

*Proof.* We write the output of the signing protocol $\sigma$ in the form $(c', s', \Phi(r, \varphi))$. By the correctness of $\mathsf{BS}[\mathsf{LF}]$, $c' = \mathcal{H}(\mu, \mathsf{F}(s') - c' \cdot \mathsf{pk})$ where $\mu = \mathsf{Randomize}(\mathsf{Commit}(m; r); \varphi)$. Thus, by the correctness of the randomizable commitment scheme, $\mathsf{Randomize}(\mathsf{Commit}(m; r); \varphi) = \mathsf{Commit}(m; \Phi(r, \varphi))$, proving the correctness of $\mathsf{CCCBS}[\mathsf{LF}]$. □

# 4 Blindness

In this section, we state and prove the theorem guaranteeing blindness of our construction given that $\mathsf{BS}[\mathsf{LF}]$ is blind.

**Theorem 4.1.** *Let $\mathcal{H}, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3, \mathcal{H}_4$ be random oracles, $\mathsf{LF}$ be a linear function family, and $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Randomize})$ be a $(t, \epsilon_{\mathsf{Hide}})$-hiding and $(t, \epsilon_{\mathsf{Rand}})$-randomizable commitment. If there exists an adversary $\mathcal{A}$ against blindness game of $\mathsf{CCCBS}[\mathsf{LF}]$ where $\mathcal{A}$ runs in time $t$, makes at most $q_{\mathcal{H}_1}, q_{\mathcal{H}_2}, q_{\mathcal{H}_3}, q_{\mathcal{H}_4}$ to $\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3, \mathcal{H}_4$ respectively, and advantage $\epsilon$, then there exists an adversary against blindness of $\mathsf{BS}[\mathsf{LF}]$ running in time $t' \approx t$ and advantage at least*

$$\frac{1}{N^2}\left(\epsilon - \frac{2(N^L + N^R) \cdot (q_{\mathcal{H}_1} + q_{\mathcal{H}_2} + q_{\mathcal{H}_3})}{2^\kappa} - 2\epsilon_{\mathsf{Rand}} - 2\epsilon_{\mathsf{Hide}}\right)$$

*where $N = \max(N^L, N^R)$, and $N^L$, $N^R$ are the number of sessions $\mathcal{A}$ selected for its left and right oracle respectively.*

*Proof idea.* Following [14], we begin by defining the event $\mathsf{Bad}$ where the adversary correctly guesses some of the generated seeds and queries the oracles using those seeds before receiving them.

We then define a reduction $\mathcal{B}$ to break the blindness of $\mathsf{BS}[\mathsf{LF}]$ using the adversary attacking the blindness of $\mathsf{CCCBS}[\mathsf{LF}]$; we only care what happens conditioned on the event $\mathsf{Bad}$ not happening. The reduction tries to guess the values $I^L, I^R$ that the adversary will pick for both left and right oracles. If the guesses are correct and $\mathsf{Bad}$ does not occur, we want the view of the adversary in our reduction will be indistinguishable from that in the actual blindness game. In order to show that it is indeed the case, we consider a series of experiments in which the $\mathsf{CCCBS}[\mathsf{LF}]$ blindness challenger get modified: first, it aborts when $\mathsf{Bad}$ happens; next, it guesses the values $I^L, I^R$ that the adversary will pick; finally, it forms the commitments $\mu_{I^L}$ and $\mu_{I_R}$ differently yet indistinguishably.

Thus, the reduction succeeds in the blindness game against the challenger running $\mathsf{BS}[\mathsf{LF}]$ with the advantage negligibly different from the adversary's. For the other case, the reduction can randomly guess the bit, so it succeeds with probability $\frac{1}{2}$. Combining everything, the reduction has non-negligible advantage of breaking the blindness of $\mathsf{BS}[\mathsf{LF}]$. □

*Proof.* Let $\mathcal{A}$ be an adversary of the blindness game of $\mathsf{CCCBS}[\mathsf{LF}]$ that succeeds with probability $\frac{1}{2} + \epsilon$.

Consider an event $\mathsf{Bad}$ where $\mathcal{A}$ queries $\mathcal{H}_1, \mathcal{H}_2$, or $\mathcal{H}_3$ with some of the generated seeds before $\mathcal{A}$ receives them. The probability of $\mathsf{Bad}$ occurring is at most $\frac{2(N^L+N^R)\cdot(q_{\mathcal{H}_1}+q_{\mathcal{H}_2}+q_{\mathcal{H}_3})}{2^\kappa}$ because there are at most $2(N^L + N^R)$ seeds independently and uniformly generated from $\{0,1\}^\kappa$ via the random oracles. Hence, the probability that $\mathcal{A}$ succeeds without $\mathsf{Bad}$ happening is at least $\frac{1}{2} + \epsilon - \frac{2(N^L+N^R)\cdot(q_{\mathcal{H}_1}+q_{\mathcal{H}_2}+q_{\mathcal{H}_3})}{2^\kappa}$.

Consider the following reduction $\mathcal{B}$ breaking the blindness of $\mathsf{BS}[\mathsf{LF}]$ using $\mathcal{A}$ as a subroutine:

1. $\mathcal{A}$'s random oracle queries for $\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3, \mathcal{H}_4$ are handled honestly, and $\mathcal{A}$'s queries for $\mathcal{H}$ are forwarded to $\mathcal{B}$'s challenger. The only exception is when $\mathsf{Bad}$ occurs, $\mathcal{B}$ aborts and outputs a random bit $b' \leftarrow \{0,1\}$.

2. $\mathcal{B}$ receives $\mathsf{pk}, m_0, m_1$ from $\mathcal{A}$ and commits to $m_0, m_1$ resulting in $\mu_0, \mu_1$ such that $\mu_i := \mathsf{Commit}(m_i; \varphi_i), \varphi_i \leftarrow \mathcal{X}$ for $i = 0, 1$. Then, forward $\mathsf{pk}, \mu_0, \mu_1$.

3. Here, we will explain what happens in the left execution of $\mathcal{B}$ with $\mathcal{A}$, and the right execution will be analogous. (Superscripts $L, R$ indicates left and right execution respectively.)

   (a) $\mathcal{B}$ generates $C^L := \mathsf{Commit}(m_0; r^L)$ such that $r^L \leftarrow \{0,1\}^\kappa$ and sends to $\mathcal{A}$

   (b) $\mathcal{A}$ replies with $N^L$. $\mathcal{B}$ initializes $i^L \leftarrow [N^L]$ and runs the user protocol honestly. Upon receiving, $R_1^L, \ldots, R_{l^L}^L$ $\mathcal{B}$ calculates $\tilde{R}_{i^L}^L$ and sends that to its left execution. Set $c_{i^L}^L$ as the response from $\mathcal{B}$'s own left oracle. For other $i \neq i^L$, run the user protocol honestly.

   (c) $\mathcal{B}$ follows the user protocol until receiving $I^L$. If $I^L \neq i^L$, $\mathcal{B}$ aborts the experiment and outputs a random bit $b' \leftarrow \{0,1\}$. Else, follow the user protocol honestly.

   (d) $\mathcal{B}$ receives $s_{i^L}^L$ from $\mathcal{A}$, and forwards it to its left oracle.

4. $\mathcal{B}$ receives $(\mu_0, s_0', c_0'), (\mu_1, s_1', c_1')$, and forwards to $\mathcal{A}$ as $(m_0, s_0', c_0', \varphi_0), (m_1, s_1', c_1', \varphi_1)$. $\mathcal{B}$ then forwards the output $b'$ from $\mathcal{A}$ to the challenger.

To analyze $\mathcal{B}$'s success probability, we will consider the following experiments conditioning on the events $i^L = I^L$ and $i^R = i^R$.

**Experiment 0:** Blindness game of $\mathsf{CCCBS}[\mathsf{LF}]$. $\mathcal{A}$'s advantage in this experiment is $\epsilon$.

**Experiment 1:** Identical to Experiment 1, except that the challenger aborts when $\mathsf{Bad}$ happens. $\mathcal{A}$'s advantage in this experiment is $\epsilon - 2(N^L + N^R) \cdot (q_{\mathcal{H}_1} + q_{\mathcal{H}_2} + q_{\mathcal{H}_3})/2^\kappa$.

**Experiment 2:** Identical to Experiment 1, except that the challenger does the same procedure as our reduction $\mathcal{B}$ by uniformly choosing the index $i^L$ and in the $i^L$-th session of the left oracle uses $\mu_{i^L} := \mathsf{Commit}(m_b; \varphi_b)$ where $\varphi_b \leftarrow \mathcal{X}$. Then, when outputting the signatures, the challenger uses $\varphi_b$ instead of $\Phi(r^L, \varphi_{i^L}^L)$.

When $i^L = I^L$, $\mathcal{A}$'s advantage in this game differs from its advantage in Experiment 1 at most $\epsilon_{\mathsf{Rand}}$, because our commitment scheme is $(t, \epsilon_{\mathsf{Rand}})$-randomizable.

**Experiment 3:** Identical to Experiment 2, except that the challenger uniformly chooses the index $i^R$ and in the $i^R$-th session of the right oracle uses $\mu_{i^R} := \mathsf{Commit}(m_{1-b}; \varphi_{1-b})$ where $\varphi_{1-b} \leftarrow \mathcal{X}$. Then, when outputting the signatures, the challenger uses $\varphi_{1-b}$ instead of $\Phi(r^R, \varphi_{i^R}^R)$.

Consider only when $i^L = I^L$ and $i^R = i^R$. By the same argument as above, $\mathcal{A}$'s advantage in this game differs from its advantage in Experiment 2 at most $\epsilon_{\mathsf{Rand}}$.

**Experiment 4:** Identical to Experiment 3, except that for both cases of $b = 0, 1$, the challenger commits $C^L := \mathsf{Commit}(m_0; r^L)$ and uses this commitment throughout the left oracle's execution.

Consider only when $i^L = I^L$ and $i^R = i^R$. $\mathcal{A}$'s advantage in this game differs from its advantage in Experiment 3 at most $\epsilon_{\mathsf{Hide}}$, because our commitment scheme is $(t, \epsilon_{\mathsf{Hide}})$-hiding. Note that in the reduction to show this claim, the reduction does not need to know the opening of $C^L$ to complete the oracle execution.

**Experiment 5:** Identical to Experiment 4, except that for both cases of $b = 0, 1$, the challenger commits $C^R := \mathsf{Commit}(m_1; r^R)$ and uses this commitment throughout the left oracle's execution.

Consider only when $i^L = I^L$ and $i^R = i^R$. With the same argument, $\mathcal{A}$'s advantage in this game differs from its advantage in Experiment 4 at most $\epsilon_{\mathsf{Hide}}$.

Hence, in Experiment 5 when $i^L = I^L$ and $i^R = I^R$, $\mathcal{A}$ succeeds with advantage at least $\epsilon - 2(N^L + N^R) \cdot (q_{\mathcal{H}_1} + q_{\mathcal{H}_2} + q_{\mathcal{H}_3})/2^\kappa - 2\epsilon_{\mathsf{Hide}} - 2\epsilon_{\mathsf{Rand}}$. Also, when $i^L = I^L$ and $i^R = I^R$, Experiment 5 is identical to the reduction $\mathcal{B}$'s interaction with $\mathcal{A}$. Therefore, if $\mathcal{A}$ succeeds while $\mathsf{Bad}$ does not happen and $i^L = I^L, i^R = I^R$, $\mathcal{B}$ succeeds in breaking the blindness of blind signature on $\mathsf{LF}$. In particular,

$$\begin{aligned}\mathbb{P}[\mathcal{B} \text{ succeeds}] &\geq \frac{1}{2}\,\mathbb{P}[I^L \neq i^L \vee I^R \neq i^R] + \mathbb{P}[\mathcal{A} \text{ succeeds in Experiment 5} \wedge I^L = i^L \wedge I^R = i^R] \\ &\geq \frac{1}{2} + \frac{1}{N^L N^R}\left(\epsilon - \frac{2(N^L + N^R) \cdot (q_{\mathcal{H}_1} + q_{\mathcal{H}_2} + q_{\mathcal{H}_3})}{2^\kappa} - 2\epsilon_{\mathsf{Rand}} - 2\epsilon_{\mathsf{Hide}}\right)\end{aligned}$$

proving the theorem. □

# 5 Unforgeability

In this section, we state and prove the theorem guaranteeing one-more unforgeability claim of $\mathsf{CCCBS}[\mathsf{LF}]$ assuming that $\mathsf{BS}[\mathsf{LF}]$ is $\lambda$-one-more unforgeable for $\lambda = \mathcal{O}(\log \kappa)$ .

**Theorem 5.1.** *Let $\mathcal{H}, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3, \mathcal{H}_4$ be random oracles, $\mathsf{LF}$ be a linear function family with $(t, \epsilon_{PRE})$-preimage resistant, and $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Randomize})$ be a $(t, \epsilon_{\mathsf{Bind}})$-binding randomizable commitment scheme. With $\ell = \mathsf{poly}(\kappa)$, suppose there exists an adversary $\mathcal{A}$ with success probability $\epsilon$ in the $\ell$-one-more unforgeability game with $\mathsf{CCCBS}[\mathsf{LF}]$ such that $\mathcal{A}$ runs in time $t$, initiates the protocol at most $p$ times, and makes at most $q_{\mathcal{H}}, q_{\mathcal{H}_1}, q_{\mathcal{H}_2}, q_{\mathcal{H}_3}, q_{\mathcal{H}_4}$ queries to $\mathcal{H}, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3, \mathcal{H}_4$ respectively.*

*Then, there exists an adversary $\mathcal{B}$ in $\lambda$-one-more unforegability game of $\mathsf{BS}[\mathsf{LF}]$, where $\lambda = 3\lceil \log p \rceil + \log \frac{2}{\epsilon}$, such that $\mathcal{B}$ runs in time $t' \approx t$, initiates the protocol at most $p$ times, makes at most $q_{\mathcal{H}}$ queries to $\mathcal{H}$, and succeeds with probability*

$$\frac{\epsilon}{2} - \frac{4q_{\mathcal{H}_1}^2 + q_{\mathcal{H}_2}^2 + q_{\mathcal{H}_4}^2 + 2p \cdot q_{\mathcal{H}_2} + p \cdot q_{\mathcal{H}_4} + p^2(p^2 + q_{\mathcal{H}})}{2^\kappa} - p \cdot \epsilon_{PRE} - \epsilon_{\mathsf{Bind}}$$

*Proof idea.* Our proof is based on a similar hybrid argument from [14]'s one-more unforgeability proof. Detailed differences can be found in Appendix C.

First, we observe that the adversary can successfully cheat only when it guesses the index $I$ correctly before committing its random seeds and challenges. (By "successful cheating", we mean that the adversary sends in a commitment not related to any of previous random oracle queries, but later the signer does not catch any inconsistency.) This results in at most $1/N$ probability of successful cheating in a protocol execution with state $N$. Then, we upper-bound the number of successful cheating the adversary $\mathcal{A}$ can achieve over the course of $\ell$ completed protocol executions to be at most $\lambda$. For the case when the adversary does not cheat, the reduction sets the values $R_1, \ldots, R_l$ and programs the random oracle $\mathcal{H}$ in a way that if the reduction is required to output

the response $s_I$ to a challenge $c_I$, the reduction can respond without using the secret key. We call the signatures from these executions *fake*. For the case that the adversary attempts to cheat, the reduction lets its challenger respond instead which makes the number of issued signatures from the challenger the same as the number of times the adversary successfully cheats.

When the adversary outputs $\ell+1$ message-signature pairs, the reduction identifies and removes all the *fake* pairs from these $\ell+1$ pairs. Lastly, the reduction sends the remaining message-signature pairs to its challenger. We will see that the number of remaining signatures is greater than the number of successful cheating. Since we established that the number of successful cheating is the same as the number of issued signatures from the challenger, the reduction wins with almost the same probability as the adversary $\mathcal{A}$. □

*Proof.* Let $\mathcal{A}$ be an adversary playing the one-more unforgeability game with the challenger running $\mathsf{CCCBS}[\mathsf{LF}]$. Suppose that $\mathcal{A}$ runs in in time $t$, initiates the protocol at most $p$ times, makes at most $q_{\mathcal{H}}, q_{\mathcal{H}_1}, q_{\mathcal{H}_2}, q_{\mathcal{H}_3}, q_{\mathcal{H}_4}$ queries to $\mathcal{H}, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3, \mathcal{H}_4$ respectively, and has success probability $\epsilon$. Let $\ell$ be the number of completed protocol sessions.

**Experiment 0:** identical to $\ell$-one-more unforgeability game. The success probability of $\mathcal{A}$ in this experiment is $\epsilon$.

We say that a value $c$ is *extractable* from a random oracle $\mathcal{H}$ if there is a query $v$ to $\mathcal{H}$ such that $\mathcal{H}(v) = c$.

**Experiment 1:** This experiment is identical to Experiment 0, except *the experiment aborts* when at least one of the following occurs in one of the signing executions:

(1) There are collisions in $\mathcal{H}_2, \mathcal{H}_4$ or some first or last $\kappa$ bits of the outputs of $\mathcal{H}_1$ are identical.

(2) $H_1$, is not extractable from $\mathcal{H}_2$ when received, but later in the same protocol execution *the signer does not abort*. (For this to occur, the adversary needs to guess $seed_{l,i}$'s which commit to $H_1$.) This event has probability at most $q_{\mathcal{H}_2}/2^{\kappa}$ to occur for each protocol execution.

(3) $H_1$ is extractable from $\mathcal{H}_2$ into $h_0, ..., h_{N+1}$, but for some $i$, $h_i$ is not extractable from $\mathcal{H}_2$; however, even when $I \neq i$ (the user needs to reveal $seed_{l,i}$), *the signer does not abort*. The event that one of $h_i$ that is not extractable occurs with probability at most $q_{\mathcal{H}_2}/2^{\kappa}$ for each protocol execution.

(4) $H_2$ is not extractable from $\mathcal{H}_4$ when received, but *the signer does not abort* later. This event has probability at most $q_{\mathcal{H}_4}/2^{\kappa}$ to occur for each protocol execution.

(5) $\mathcal{A}$ outputs two signatures $(m, s, c, \varphi), (m', s, c, \varphi')$ such that $m \neq m'$ but $\mathsf{Commit}(m; \varphi) = \mathsf{Commit}(m'; \varphi')$. This event has probability at most $\epsilon_{\mathsf{Bind}}$ of happening, which is negligible.

Overall in Experiment 1, $\mathcal{A}$ succeeds with probability at least

$$\epsilon - \left(4q_{\mathcal{H}_1}^2 + q_{\mathcal{H}_2}^2 + q_{\mathcal{H}_4}^2 + 2p \cdot q_{\mathcal{H}_2} + p \cdot q_{\mathcal{H}_4}\right)/2^{\kappa} - \epsilon_{\mathsf{Bind}}$$

The purpose of the limitations in this experiment is to (1) ensure that each hash value corresponds to a unique query, (2,3,4) rule out the possibility of the adversary cheating by not sending correct hashes but was able to guess the input to get the correct hashes, and (5) prohibit the adversary in outputting two distinct message-signature pairs that are identical after computing the commitment of the message.

**Definition 5.2** (Successful Cheating)**.** *The adversary $\mathcal{A}$ successfully cheats if* ***the signer does not abort****, but one of the following occurs:*

*(1) $H_1$, when received, is extractable from $\mathcal{H}_2$ into $h_0, ..., h_{N+1}$, but for some $i$, $h_i$, is not extractable from $\mathcal{H}_2$ into $seed_{l,i}$, or*

*(2) Although $H_1$ is extractable from $\mathcal{H}_2$ into $seed_{l,i}$'s, and the challenger can derive $\alpha_I, \beta_I, \varphi_I$ generated from $seed_{l,I}$, $c_I \neq \mathcal{H}\left(\mu_I = \mathsf{Randomize}(C; \varphi_I), \tilde{R}_I + \mathsf{F}(\alpha_I) + \beta_I \cdot \mathsf{pk}\right) + \beta_I$.*

With this definition, in Experiment 1, $\mathcal{A}$ can successfully cheat only if it guesses $I$ correctly and commits the wrong seed or the wrong $c_I$. Hence, in a signing session, $\mathcal{A}$ has at most $\frac{1}{N}$ probability of successfully cheating.

Next, we will upper-bound the expected number of times $\mathcal{A}$ successfully cheats in Experiment 1. Consider an integer $k$ such that at protocol execution $p$, $N = 2^k - 2$. By how the protocol increments $N$ and for large $k$,

$$p \geq \sum_{i=2}^{k-1} 2^i - 2 = 2^k - 2k \geq 2^{k-1}$$

so $k \leq \lceil \log p \rceil + 1$. Thus, the expected number of cheating in Experiment 1 is upper-bounded by

$$\sum_{i=2}^{k} \sum_{j=1}^{2^i-2} \frac{1}{2^i - 2} \leq \sum_{i=2}^{\lceil \log p \rceil + 1} \sum_{j=1}^{2^i-2} \frac{1}{2^i - 2} = \lceil \log p \rceil$$

**Experiment 2:** This experiment is identical to Experiment 1, except *the experiment aborts* when the adversary successfully cheats more than $\lambda = 3\lceil \log p \rceil + \log \frac{2}{\epsilon}$ times. The purpose of this step is to limit the number of cheating happening in the experiment.

We can upper-bound the probability that $\mathcal{A}$ cheats more than $\lambda$ times in Experiment 1 to $\epsilon/2$:

Let $X$ be a random variable for the number of times $\mathcal{A}$ successfully cheats in Experiment 1 and $k$ be the value such that at protocol execution $p$, $N = 2^k - 2$. By the analysis above, $k \leq \lceil \log p \rceil + 1$. The random variable $X$ is then bounded by the following sum of iid Bernoulli random variables i.e.

$$X \leq \sum_{i=2}^{\lceil \log p \rceil + 1} \sum_{j=1}^{2^i-2} \text{Bernoulli}\left(\frac{1}{2^i - 2}\right)$$

Let $Y$ be the sum of these iid Bernoulli random variables and $\mu_Y = \mathbb{E}[Y] = \lceil \log p \rceil$.

$$\begin{aligned}
\mathbb{P}[X \geq \lambda] &\leq \mathbb{P}[Y \geq \lambda] \\
&= \mathbb{P}[Y \geq \mu_Y(1 + (\lambda/\mu_Y - 1))] \\
&\leq \exp\left(-\frac{\mu_Y(\lambda/\mu_Y - 1)^2}{2 + (\lambda/\mu_Y - 1)}\right) \ ; \text{ By Chernoff's Bound}
\end{aligned}$$

With $x^2/(2+x) \geq x - 2$ for all $x \geq 0$,

$$\mathbb{P}[X \geq \lambda] \leq \exp\left(-\mu_Y\left(\frac{\lambda}{\mu_Y} - 3\right)\right) = \exp\left(3\lceil \log p \rceil - 3\lceil \log p \rceil - \log \frac{2}{\epsilon}\right) = \frac{\epsilon}{2}$$

Thus, $\mathcal{A}$ successfully cheats in Experiment 1 with probability at most $\epsilon/2$. Therefore, $\mathcal{A}$'s success probability in Experiment 2 is at least $\epsilon/2 - \left(4q_{\mathcal{H}_1}^2 + q_{\mathcal{H}_2}^2 + q_{\mathcal{H}_4}^2 + 2p \cdot q_{\mathcal{H}_2} + p \cdot q_{\mathcal{H}_4}\right)/2^\kappa - \epsilon_{\mathsf{Bind}}$.

**Experiment 3:** This experiment is identical to Experiment 2 except that, after the signer receives $H_1$ in each protocol execution, the experiment does the following:

- The challenger randomizes $j \leftarrow [N]$.
- If $H_1$ is extractable from $\mathcal{H}_2$ into $h_0, ..., h_{N+1}$ and $h_j$ is extractable from $\mathcal{H}_2$ into $seed_{l,j}$:
  (a) $C_j \leftarrow \mathcal{S}$
  (b) Randomly select $i'$ among the 1-bits of $j$.
  (c) $r_{i'} \leftarrow \mathcal{D}, R_{i'} := \mathsf{F}(r_{i'}) + C_j \cdot (-\mathsf{pk})$.
  (d) After initializing other $R_i$'s, calculate $\alpha_j, \beta_j, \varphi_j, \mu_j, \tilde{R}'_j$ using the extracted $seed_{l,j}$. If $\mathcal{H}(\mu_j, \tilde{R}'_j)$ is already initialized, *the experiment aborts.* If not, set $\mathcal{H}(\mu_j, \tilde{R}'_j) := C_j - \beta_j$.
  (e) Later when $s_I$ needs to be calculated where $i'$-th bit of $I$ is 1, use $r_{i'} + C_j \cdot (-\mathsf{sk})$ as the discrete log witness of $R_{i'}$.

  **Note:** We call the above session $j$ that has $\mathcal{H}$ defined in this way as a *programmed session.*

  If $H_1$ or $h_j$ is not extractable from $\mathcal{H}_2$, set $C_j := \perp$.
- After initializing $R_i$ for each $i \in [l] \setminus \{i'\}$, for each $k \in [N] \setminus \{j\}$, if $H_1$ is not extractable from $\mathcal{H}_2$ , or it is extractable from $\mathcal{H}_2$ into $h_0, ..., h_{N+1}$, but $h_k$ is not extractable from $\mathcal{H}_2$, set $C_k := \perp$.

  Otherwise, $h_k$ is extractable into $seed_{l,k}$, so the experiment checks if $\mathcal{H}$ is initialized at $(\mu_k, \tilde{R}'_k)$. If it is, *the experiment aborts.* Else, uniformly randomly set $\mathcal{H}(\mu_k, \tilde{R}'_k)$ and let $C_k := \mathcal{H}(\mu_k, \tilde{R}'_k) + \beta_k$.
- Send $R_1, ..., R_l$ to $\mathcal{A}$ after all of the above is done.

Ignoring the aborts, we claim that the view of the adversary $\mathcal{A}$ is identical to its view in Experiment 2. To see this, first, the outputs of the random oracle $\mathcal{H}$ are still uniformly distributed in $\mathcal{S}$. Moreover, the distribution of $R_1, ..., R_l$ stays the same because the distribution of $R_{i'} = \mathsf{F}(r_{i'}) + C_j \cdot (-\mathsf{pk}) = \mathsf{F}(r_{i'} + C_j \cdot (-\mathsf{sk}))$ is the same as $\mathsf{F}(r_{i'})$ given that $r_{i'}$ is uniformly generated. Furthermore, for the programmed session $j$, the joint distribution of $(\tilde{R}'_j, c_j, s_j)$ remains the same because $\tilde{R}'_j = \mathsf{F}(s_j) + c_j \cdot \mathsf{pk}$ in both experiments.

In this experiment, the change of $\mathcal{A}$'s success probability only comes from the experiment aborting when it cannot program the oracle $\mathcal{H}$ at some $(\mu, R)$. For $\mathcal{H}$ to be initialized at some $(\mu, R)$ point, it needs to be queried from $\mathcal{A}$ or programmed by the experiment. Hence, we can consider the following three cases where $(\mu, R)$ has been initialized:

(1) $(\mu, R)$ has been queried by $\mathcal{A}$. The experiment tries to program the oracle with $R = \sum_{i \in S_k} R_i + \mathsf{F}(\alpha_k) + \beta_k \cdot \mathsf{pk}$ for some $k \in [N]$, and $R_i = \mathsf{F}(r_i)$ is uniform in $\mathcal{R}$ as $r_i$ is uniform in $\mathcal{D}$. Also, the value of $R$ that $\mathcal{A}$ queried the random oracle is independent of $R_i$ because the experiment has not revealed $R_i$ to the adversary yet. Thus, this happens with probability at most $1/2^\kappa$.

(2) $(\mu, R)$ has been programmed by the experiment in another protocol execution. The same argument from above applies as $R_i$ is independent from other values from a different protocol execution. Hence, this happens with probability at most $1/2^\kappa$.

(3) $(\mu, R)$ has been programmed by the experiment in the same protocol execution. Then,

$$R = \sum_{i \in S_{k_1}} R_i + \mathsf{F}(\alpha_{k_1}) + \beta_{k_1} \cdot \mathsf{pk} = \sum_{i \in S_{k_2}} R_i + \mathsf{F}(\alpha_{k_2}) + \beta_{k_2} \cdot \mathsf{pk}$$

where $S_{k_1} \neq S_{k_2}$. Thus, there is some $R_i = \mathsf{F}(r_i)$ such that $i$ is in only one of $S_{k_1}, S_{k_2}$. Thus, $R_i$ is independent from any other values involved in the event. By the same argument, this happens with probability at most $1/2^\kappa$.

Thus, $\mathcal{A}$'s success probability is reduced by at most $p^2(p^2 + q_\mathcal{H})/2^\kappa$ where $p^2$ upper-bounds the number of times the challenger tries to program $\mathcal{H}$ and $p^2 + q_\mathcal{H}$ upper-bounds the number of times $\mathcal{H}$ is programmed or queried. Therefore, $\mathcal{A}$'s success probability in this experiment is $\epsilon/2 - \left(4q_{\mathcal{H}_1}^2 + q_{\mathcal{H}_2}^2 + q_{\mathcal{H}_4}^2 + 2p \cdot q_{\mathcal{H}_2} + p \cdot q_{\mathcal{H}_4} - p^2(p^2 + q_\mathcal{H})\right)/2^\kappa - \epsilon_{\mathsf{Bind}}$.

**Experiment 4:** This experiment is identical to Experiment 3, except how the signer generates random index $I$: After receiving $H_2$, if $H_2$ is extractable from $\mathcal{H}_4$ into $c_1, ..., c_N$ and for all $i \in [N]$, $c_i = C_i$ i.e. there is no cheating from the adversary, set $I := j$ where $j$ is the value defined in Experiment 3. Otherwise, if $H_2$ is not extractable or extractable from $\mathcal{H}_4$ into $c_1, ..., c_N$, but $c_i \neq C_i$ for some $i \in [N]$, set $I := (N+1) \oplus j = (2^l - 1) \oplus j$ which is the bitwise complement of $j$.

Since $j$ is uniformly randomly generated, $I$'s distribution remains uniform in both cases. Hence, the success probability of $\mathcal{A}$ remains the same as Experiment 3.

Then, we define new definitions regarding the programmed sessions and the signatures related to them.

**Definition 5.3** (Completed Programmed Sessions)**.** *A programmed session $j$ is completed if $I = j$ and the signer does not abort the signing protocol.*

**Definition 5.4** (Fake signature)**.** *A valid signature $\sigma = (m, c', s', \varphi')$ is a fake signature if, with $R' := \mathsf{F}(s') - c' \cdot \mathsf{pk}$ and $\mu := \mathsf{Commit}(m; \varphi')$, there is a programmed session(not necessarily a completed one) that $\mathcal{H}$ is programmed at $(\mu, R')$.*

**Claim 5.5.** *In Experiment 4, if $\mathsf{LF}$ is $(t, \epsilon_{PRE})$-preimage resistant, then the probability that any adversary $\mathcal{A}$ succeeds and outputs more fake signatures (Definition 5.4) than the number of completed programmed sessions is at most $p \cdot \epsilon_{PRE}$.*

*Proof.* Let $\mathcal{A}$ be an adversary in Experiment 4, and $E$ be the event that $\mathcal{A}$ succeeds and outputs more fake signatures than the number of completed programmed sessions. From Experiment 1, $\mathcal{A}$ cannot output two different signatures $(m_0, c', s', \varphi_0), (m_1, c', s', \varphi_1)$ such that $\mu = \mathsf{Commit}(m_0; \varphi_0) = \mathsf{Commit}(m_1; \varphi_1)$.

Then, we will construct a reduction $\mathcal{B}'$ using $\mathcal{A}$ as a subroutine to break preimage resistance of $\mathsf{LF}$.

1. $\mathcal{B}'$ receives the input $(\mathsf{params}, R)$ from the challenger of preimage resistant game for $\mathsf{LF}$. $\mathcal{B}'$ then randomizes $k \leftarrow [p]$. In the $k^{th}$ execution of the signing protocol, instead of setting $R_{i'} := \mathsf{F}(r_{i'}) + C_j \cdot (-\mathsf{pk})$, we set $R_{i'} := R + C_j \cdot (-\mathsf{pk})$. Recall that $i'$-th bit of programmed session $j$ is 1, so $\tilde{R}_j$ will compose of $R$. If $\mathcal{B}'$ needs to calculate $s_j$, it will abort.

2. Later, if $\mathcal{A}$ succeeds and outputs more fake signatures than the number of *completed* programmed sessions, and the first fake signature which does not correspond to a *completed* programmed session is from the $k^{th}$ execution, $\mathcal{B}'$ can compute the preimage of $R$ as follows: Let $\sigma = (m, c', s', \varphi), R' = \mathsf{F}(s') - c' \cdot \mathsf{pk}$, and $r' = \sum_{i \neq i' : i \in S_j} r_i$.

$$\begin{aligned} R' &= \tilde{R}'_j \\ \mathsf{F}(s') - c' \cdot \mathsf{pk} &= R + C_j \cdot (-\mathsf{pk}) + \mathsf{F}(\alpha_j) + \beta_j \cdot \mathsf{pk} + \sum_{i \neq i' : i \in S_j} \mathsf{F}(r_i) \\ \mathsf{F}(s') &= R + \mathsf{F}(\alpha) + c' \cdot \mathsf{pk} + C_j \cdot (-\mathsf{pk}) + (C_j - c') \cdot \mathsf{pk} + \mathsf{F}\left(r'\right) \\ &= R + \mathsf{F}\left(\alpha + r'\right) + c' \cdot \mathsf{pk} + C_j \cdot (-\mathsf{pk}) + C_j \cdot \mathsf{pk} - c' \cdot \mathsf{pk} + \mathsf{F}(\Psi(\mathsf{pk}, C_j, -c')) \\ R &= \mathsf{F}\left(s' - \alpha - \Psi(\mathsf{pk}, C_j, -c') - r'\right) \end{aligned}$$

giving us the preimage of $R$.

Therefore, $\mathcal{B}'$ can win the preimage resistant game with $1/p$ of $\mathbb{P}[E]$. Hence, with $\mathsf{LF}$ being $(t, \epsilon_{\mathsf{PRE}})$-preimage resistant, $\mathbb{P}[E] \leq p \cdot \epsilon_{\mathsf{PRE}}$. □

**Experiment 5:** This experiment is identical to Experiment 4 except that, after the adversary outputs the signatures, *the experiment aborts* if there are more more fake signatures than the number of completed programmed sessions.

From Claim 5.5, the success probability of $\mathcal{A}$ in Experiment 5 is at least

$$\frac{\epsilon}{2} - \frac{4q_{\mathcal{H}_1}^2 + q_{\mathcal{H}_2}^2 + q_{\mathcal{H}_4}^2 + 2p \cdot q_{\mathcal{H}_2} + p \cdot q_{\mathcal{H}_4} + p^2(p^2 + q_{\mathcal{H}})}{2^\kappa} - p \cdot \epsilon_{\mathsf{PRE}} - \epsilon_{\mathsf{Bind}}$$

Lastly, we will construct a reduction $\mathcal{B}$ trying to break $\lambda$-one-more unforgeability of $\mathsf{BS}[\mathsf{LF}]$ where $\lambda = 3\lceil \log p \rceil + \log \frac{2}{\epsilon}$, simulating Experiment 5, and using adversary $\mathcal{A}$ as a subroutine

The reduction $\mathcal{B}$ is defined as follows:

1. For the random oracle queries not for $\mathcal{H}$, $\mathcal{B}$ outputs a random value normally. For queries to $\mathcal{H}$, $\mathcal{B}$ forwards the query to its challenger and sends back the output from the challenger to $\mathcal{A}$. However, on the programmed sessions, $\mathcal{B}$ will answer $\mathcal{A}$'s queries for the programmed sessions by itself by how it programs $\mathcal{H}$ in Experiment 3.

2. To start the game, $\mathcal{B}$ receives the public key $\mathsf{pk}$ from its challenger and forwards it to $\mathcal{A}$.

3. For each signing protocol execution from $\mathcal{A}$, $\mathcal{B}$ replicates the challenger in Experiment 5 and additionally does the following:

   - After $\mathcal{A}$ initiates a protocol execution, start a signing session with its challenger and receive $R^*$. After initializing $j \leftarrow [N]$ and $R_{i'}$ according to Experiment 5, pick another $i''$ such that $i''$-th bit of $j$ is 0 and let $R_{i''} := R^*$.

     Then, $\mathcal{B}$ interacts with $\mathcal{A}$ by following the protocol of $\mathcal{A}$'s challenger in Experiment 5.
   - When the protocol requires $\mathcal{B}$ to send $s_I$ to $\mathcal{A}$, $\mathcal{B}$'s behavior will depend on the value of $I$.

     If $I = j$, $\mathcal{B}$ sends $s_I := \sum_{k \in S_I} r_k + C_I \cdot (-\mathsf{sk}) + c_I \cdot \mathsf{sk} = \sum_{k \in S_I} r_k$ which doesn't require $\mathcal{B}$ to know the secret key because $c_I = C_I$ by how $\mathcal{H}$ is programmed in Experiment 3 and how $I$ is selected in Experiment 4.

For the case that $I = (N+1) \oplus j$, $\mathcal{B}$ forwards $c_I$ to its challenger. Upon receiving $s^*$ from its challenger, $\mathcal{B}$ sends $s_I := s^* + \sum_{k \in S_I \setminus \{i''\}} r_k$ to $\mathcal{A}$.

4. After $\mathcal{A}$ sends $\ell + 1$ valid signatures to $\mathcal{B}$, $\mathcal{B}$ identifies and removes all the fake signatures from the $\ell + 1$ signatures. Then, $\mathcal{B}$ forwards the remaining signatures to the challenger.

Using the analysis above(see Experiment 1), we know all the $\ell + 1$ signatures of the form $(m, c', s', \varphi')$ exist no two signatures sharing the same $\mu = \mathsf{Commit}(m; \varphi')$. Also, the number of fake signatures is at most the number of completed programmed sessions which is upper-bounded by $\ell - \mathsf{cheat}$ where $\mathsf{cheat}$ is the number of times $\mathcal{A}$ successfully cheats. Therefore, the number of remaining non-fake signatures is at least $\mathsf{cheat} + 1$. As the number of issued signatures from the challenger is equal to the number of times $\mathcal{A}$ successfully cheats, $\mathcal{B}$ outputs more signatures than is issued.

Since $\mathcal{B}$ is simulating Experiment 5, the the number of times $\mathcal{A}$ successfully cheats is upper-bounded by $\lambda = \lceil \log p \rceil + \log \frac{2}{\epsilon}$. Therefore, $\mathcal{B}$ successfully breaks $\lambda$-one-more unforgeability of $\mathsf{BS}[\mathsf{LF}]$ with the same probability as $\mathcal{A}$'s success probability in Experiment 5, proving the theorem. □


### Acknowledgements

Anna Lysyanskaya acknowledges support from the Facebook Faculty Research Award program.


## References


[1] Michel Abdalla, Chanathip Namprempre, and Gregory Neven. On the (im)possibility of blind message authentication codes. In David Pointcheval, editor, *CT-RSA 2006*, volume 3860 of *LNCS*, pages 262–279. Springer, Heidelberg, February 2006.

[2] Masayuki Abe and Tatsuaki Okamoto. Provably secure partially blind signatures. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 271–286. Springer, Heidelberg, August 2000.

[3] Prabhanjan Ananth, Apoorvaa Deshpande, Yael Tauman Kalai, and Anna Lysyanskaya. Fully homomorphic NIZK and NIWI proofs. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 356–385. Springer, Heidelberg, December 2019.

[4] Foteini Baldimtsi and Anna Lysyanskaya. Anonymous credentials light. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 1087–1098. ACM Press, November 2013.

[5] Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. On the (in)security of ROS. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 33–53. Springer, Heidelberg, October 2021.

[6] Stefan Brands. Untraceable off-line cash in wallets with observers (extended abstract). In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 302–318. Springer, Heidelberg, August 1994.

[7] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO'82*, pages 199–203. Plenum Press, New York, USA, 1982.

[8] Ivan Damgård and Eiichiro Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In Yuliang Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 125–142. Springer, Heidelberg, December 2002.

[9] Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 16–30. Springer, Heidelberg, August 1997.

[10] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, January 2007.

[11] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th FOCS*, pages 464–479. IEEE Computer Society Press, October 1984.

[12] Eduard Hauck, Eike Kiltz, and Julian Loss. A modular treatment of blind signatures from identification schemes. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 345–375. Springer, Heidelberg, May 2019.

[13] Ari Juels, Michael Luby, and Rafail Ostrovsky. Security of blind digital signatures (extended abstract). In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 150–164. Springer, Heidelberg, August 1997.

[14] Jonathan Katz, Julian Loss, and Michael Rosenberg. Boosting the security of blind signature schemes. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021*, pages 468–492, Cham, 2021. Springer International Publishing.

[15] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 31–53. Springer, Heidelberg, August 1993.

[16] Christian Paquin and Greg Zaverucha. U-prove cryptographic specification v1.1 (revision 3), December 2013. Released under the Open Specification Promise (http://www.microsoft.com/openspecifications/en/us/programs/osp/default.aspx).

[17] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Heidelberg, August 1992.

[18] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, June 2000.

[19] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014.

[20] Claus-Peter Schnorr. Security of blind discrete log signatures against interactive attacks. In Sihan Qing, Tatsuaki Okamoto, and Jianying Zhou, editors, *ICICS 01*, volume 2229 of *LNCS*, pages 1–12. Springer, Heidelberg, November 2001.

[21] Dominique Schröder and Dominique Unruh. Security of blind signatures revisited. *Journal of Cryptology*, 30(2):470–494, April 2017.

[22] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities "honest or bust" with decentralized witness cosigning. In *2016 IEEE Symposium on Security and Privacy*, pages 526–545. IEEE Computer Society Press, May 2016.

[23] Alexandros Zacharakis, Panagiotis Grontas, and Aris Pagourtzis. Conditional blind signatures. Cryptology ePrint Archive, Report 2017/682, 2017. https://ia.cr/2017/682.

# A Formal Definitions to BS[LF] and CCBS[LF]

**Definition A.1** (Blind Signature from Linear Function Family)**.** *Let* $\mathsf{LF}$ *be a linear function family and* $\mathcal{H}$ *be a hash function modeled as a random oracle. Define the blind signature from linear function family* $\mathsf{BS[LF]}$ *as the tuple of algorithms* $(\mathsf{PGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{User}, \mathsf{Verify})$*:*

$\mathsf{PGen}(1^\kappa)$ *runs* $\mathsf{params} \leftarrow \mathsf{LF.PGen}(1^\kappa)$ *and outputs* $\mathsf{params}$.

$\mathsf{KeyGen}(1^k)$ *randomly picks a* $\mathsf{sk} \leftarrow \mathcal{D}$*, computes* $\mathsf{pk} := \mathsf{F}(\mathsf{sk})$*, and outputs* $(\mathsf{sk}, \mathsf{pk})$ *as the secret and public key pair.*

***Signing protocol*** *between the signer* $\mathsf{Sign}(\mathsf{sk}, \mathsf{pk})$ *and the user* $\mathsf{User}(\mathsf{pk}, m)$ *(outlined in Figure 1) is defined as follows:*

*(1) The signer uniformly picks* $r \leftarrow \mathcal{D}$*, computes* $R := \mathsf{F}(r)$*, and sends* $R$ *to the user.*

*(2) The user uniformly randomly generates* $\alpha \leftarrow \mathcal{D}, \beta \leftarrow \mathcal{S}$*, computes* $R' := R + \mathsf{F}(\alpha) + \beta \cdot \mathsf{pk}$*, queries* $c' := \mathcal{H}(m, R')$*, and sends* $c := c' + \beta$ *to the signer.*

*(3) The signer computes and sends* $s := r + c \cdot \mathsf{sk}$ *to the user.*

*(4) The user checks whether* $\mathsf{F}(s) = R + c \cdot \mathsf{pk}$ *and aborts if the check does not pass. Lastly, it outputs the signature* $\sigma := (c', s' := s + \alpha + \Psi(\mathsf{pk}, c, -c'))$.

$\mathsf{Verify}(\mathsf{pk}, m, \sigma)$ *takes as input the public key* $\mathsf{pk}$ *and the message-signature pair* $(m, \sigma = (c', s'))$*. Then, it accepts if* $c' = \mathcal{H}(m, \mathsf{F}(s') - c' \cdot \mathsf{pk})$ *and rejects otherwise.*

**Definition A.2** (Cut-and-Choose Blind Signature)**.** *Let* $\mathsf{BS[LF]}$ *be a blind signature scheme from linear function family* $\mathsf{LF}$ *with a hash function* $\mathcal{H}$*, and* $\mathcal{H}'$ *be a hash function modeled as a random oracle. Define the cut-and-choose blind signature* $\mathsf{CCBS[LF]}$ *as the tuple of algorithms* $(\mathsf{PGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{User}, \mathsf{Verify})$.

***Parameter generation:*** *remains that same as* $\mathsf{BS[LF].PGen}$.

***Key generation algorithm:*** *remains the same as* $\mathsf{BS[LF].KeyGen}$ *except that it also outputs the state* $N := 1$.

***Signing protocol*** *between the signer* $\mathsf{Sign}(\mathsf{sk}, \mathsf{pk})$ *and the user* $\mathsf{User}(\mathsf{pk}, m)$ *is defined as follows:*

*(1) The signer atomically increments its state* $N$ *and sends* $N$ *to the user.*

(2) *For each $i \in [N]$, the user prepares its randomness, commits the message $m$ to $\mu_i$ with randomness $\varphi_i$, and commits all the randomness and the commitment of the message to $\mathsf{com}_i$ with randomness $\gamma_i$. The user then sends $\mathsf{com}_1, \ldots, \mathsf{com}_N$ to to signer.*

(3) *For each $i \in [N]$, the signer uniformly picks $r_i \leftarrow \mathcal{D}$, computes $R_i := \mathsf{F}(r_i)$, and forwards $R_1, \ldots, R_N$ to the user.*

(4) *For each $i \in [N]$, the user blinds $R_i$ to $R'_i$, queries the oracle $\mathcal{H}$ getting $c'_i$, and sends $c_1, \ldots, c_N$ where $c_i := c_i + \beta$ to the signer.*

(5) *The signer uniformly picks $I \leftarrow [N]$ and sends this index to the user. The user then opens each commitment $\mathsf{com}_i$ for $i \neq I$.*

(6) *The signer verifies the openings and the challenges $c_i$'s. If these values do not verify, the signer aborts. Then, the signer computes and sends $s_I := r_I + c_I \cdot \mathsf{sk}$ to the user.*

(7) *The user checks if $\mathsf{F}(s_I) = R_I + c_I \cdot \mathsf{pk}$ and aborts if it is not. Lastly, it computes $s'_I := s_I + \alpha_I + \Psi(\mathsf{pk}, c_I, -c'_I)$ and outputs the signature $\sigma := (c'_I, s'_I, \varphi_I)$*

***Verification:*** $\mathsf{Verify}(\mathsf{pk}, m, \sigma)$ *takes as input the message-signature pair $(m, \sigma = (c', s', \varphi))$, computes $\mu := \mathcal{H}'(m, \varphi)$, and accepts if and only if $c' = \mathcal{H}(\mu, \mathsf{F}(s') - c' \cdot \mathsf{pk})$.*

# B Differences between CCBS[LF] and CCCBS[LF]

1. The state information $N$ that dictates how many sessions of BS[LF] the user creates was slightly different. In CCBS[LF], it increased by one each new protocol execution, and it was just a counter that counted protocol executions. In CCCBS[LF], we have $N$ and a separate counter ctr; the counter ctr counts the number of executions using the current $N$ so far; $N$ is an upper bound on ctr that (basically) doubles each time that ctr catches up to it.

2. In CCBS[LF], the signature recipient creates state information for $N$ blind signature sessions of the original BS[LF]; the randomness needed for each session $i$ is obtained from a separate seed $seed_i$. Here, we have a master seed value that generates $seed_i$ for each $i$; the resulting seeds are organized in a binary tree so that just $\log N$ values are sufficient to reconstruct all the seeds other than $seed_I$. As a result, we reduce the communication complexity from the user to the signer from $N$ seeds to just $\log N$.

   The user must calculate $h_i = \mathcal{H}_2(seed_i)$ for each resulting seed; this step is needed for the proof of security in the random oracle model. The user sends $H_1 = \mathcal{H}_2(h_1, \ldots, h_N)$ to the signer; once the signer picks the index $I$, the user responds with enough information that $h_i$ for $i \neq I$ can be computed locally by the signer; so the only additional value the user must provide so the signer can verify $H_1$ is $h_I$.

3. Instead of generating $N$ separate and unrelated first messages $R_1 = \mathsf{F}(r_1), \ldots, R_N = \mathsf{F}(r_N)$ to send to the user, the signer generates $l = \log N$ of them, $\mathsf{F}(r_1), \ldots, \mathsf{F}(r_l)$; each $i$, $1 \leq i \leq N$, can be viewed as an encoding of a non-empty proper subset $S_i \subsetneq [l]$, inducing $\tilde{R}_i = \sum_{j \in S_i} \mathsf{F}(r_j)$. The user can compute $\tilde{R}_i$ locally from $R_1, \ldots, R_l$ for any $i$. As a result, we reduce the communication complexity from $N$ group elements from the signer to the user to just $\log N$.

4. Since the user will ultimately have to show to the signer all the state information required for all but one of the $N$ sessions, they will have to reveal a commitment to the message used

in each of them. In CCBS[LF], the user sends all but one of these commitments to the signer, which requires $\Omega(N)$ communication complexity. Here, we need to compress the information; thus we have the user send just one commitment to the message; this commitment $C$ is formed using a randomizable commitment scheme. From $seed_i$, the user obtains randomness $\varphi_i$ that randomizes $C$ into an unlinkable commitment $\mu_i$. This reduces the communication complexity for communicating commitments to messages from $\Omega(N)$ bits to $O(1)$ commitments.

5. In the challenge step of CCBS[LF], the user sent challenges $c_1, \ldots, c_N$ to the signer; each $c_i$ was the challenge for the $i^{th}$ session of the BS[LF] protocol. To reduce the communication complexity, our protocol hashes these values down to just one value $H_2 = \mathcal{H}_4(c_1, \ldots, c_N)$. After the signer has picked the specific index $I$, the user supplies the information from which the signer derives $seed_i$ for $i \neq I$, and $c_i$ is computed from $seed_i$. Therefore, in the challenge step, the user will only need to send $c_I$ to ensure that the signer has all the information needed to verify $H_2$.

# C Differences between CCBS[LF] and CCCBS[LF] unforgeability proof

In our proof in Section 5, we follow the same hybrid argument as in [14]'s proof. Below, we list the differences between the two proofs.

1. **Experiment 1**. With more random oracles introduced into CCCBS[LF], we define our Experiment 1 slightly different from [14]'s. In the original proof, the experiment aborts when there is a collision in $\mathcal{H}'$ or when some commitment of randomness $\mathsf{com}_i$ is not extractable from $\mathcal{H}'$ for some $i \neq I$, but later the signer does not abort.

   In our proof, the experiment still aborts when there are collisions or when some seed commitments are not extractable but the signer doesn't abort. However, due to CCCBS[LF] not sending the challenges $c_i$'s but sending the commitment of them instead, our experiment 1 also aborts when this commitment $H_2$ is not extractable. Additionally, the experiment aborts when the adversary outputs two signatures $(m, s, c, \varphi), (m', s, c, \varphi')$ with $m \neq m'$ but $\mathsf{Commit}(m; \varphi) = \mathsf{Commit}(m', \varphi')$.

2. **Definition of successful Cheating**. Our definition of successful cheating is almost analogous to the original proof. In the original proof, successful cheating is defined as an event when the signer does not abort but (1) some commitment $\mathsf{com}_i$ sent by $\mathcal{A}$ in that execution was not extractable or (2) for some $i$, the commitment $\mathsf{com}_i$ sent in that execution was extractable with associated values $\alpha_i, \beta_i, \mu_i$, but $c_i \neq \mathcal{H}(\mu_i, R_i + F(\alpha_i) + \beta_i \cdot \mathsf{pk}) + \beta_i$ (where $R_i$ is the value sent by the signer in the corresponding session). In our proof, we leave (1) essentially the same about $H_1$, but for (2), we only consider $c_I$ as the user in CCCBS[LF] does not send any other $c_i$'s but the commitment of them as $H_2$.

3. **Experiment 2**. The only difference in this experiment is the value of $\lambda$ due to the difference in incrementing $N$ in the two constructions.

4. **Experiment 3**. In the original proof's experiment 3, they introduced the programmed session $j$ with $j \leftarrow [N]$ where they set $R_j := F(r_j) + C_j \cdot (-\mathsf{pk})$ with $C_j \leftarrow \mathcal{S}$ and program the oracle $\mathcal{H}(\mu_j, R'_j)$ with $C_j - \beta_j$. Due to the signing protocol of CCCBS[LF] only sending $R_1, ..., R_l$

where $l \approx \log N$, we cannot use the same trick. Here, the experiment pick the session $j \leftarrow [N]$ and $i' \leftarrow S_j$.(As a reminder, $S_j$ is defined as the set of indices where $j$ has bit 1 in such index.) Then, the experiment uniformly samples $r_{i'} \leftarrow \mathcal{D}, C_j \leftarrow \mathcal{S}$ and set $R_{i'} := \mathsf{F}(r_{i'}) + C_j \cdot (-\mathsf{pk})$. Later when the experiment initialized all $R_i$, we program $\mathcal{H}(\mu_j, \tilde{R}'_j) := C_j - \beta_j$, before sending the $R_i$'s.

5. **Experiment 4**. In this experiment, instead of setting $I := j + 1$ when the adversary cheats, the experiment instead set $I := (2^l - 1) \oplus j$ as bitwise complement of $j$. With $N = 2^l - 2$, $I$ is still uniformly distributed in $[N]$ for both cases as $j \leftarrow [N]$.

6. **Experiment 5**. Both our proof and the original proof have identical definition for experiment 5.

7. **Reduction**. In the reduction, the original proof the reduction sets $R_{j+1} := R^*$ where $R^*$ is the value received from the first round of a signing session with the reduction's signing oracle and $j$ is the programmed session. In $\mathsf{CCCBS}[\mathsf{LF}]$, we cannot set $\tilde{R}'_{j+1}$ directly, so we have to pick an index $i''$ where $i''$-th bit of $\bar{j}$ is 1 to set $R_{i''} := R^*$.

   Because of this reduction's setting and how we deal with programmed sessions in experiment 3, we need any $j \leftarrow [N]$ to have at least one 1-bit and 0-bit and that $\bar{j}$ is uniformly distributed in $N$. Hence, our $N$ is selected to always be of the form $2^l - 2$.