PI-Cut-Choo and Friends: Compact Blind Signatures via Parallel Instance Cut-and-Choose and More

Rutchathon Chairattana-Apirom*[2], Lucjan Hanzlik[1], Julian Loss[1], Anna Lysyanskaya[2], and Benedikt Wagner*[1]

[1] CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
{hanzlik,loss,benedikt.wagner}@cispa.de
[2] Brown University,
Providence RI 02906, USA
{rutchathon_chairattana-apirom,anna_lysyanskaya}brown.edu

Abstract. Blind signature schemes are one of the best-studied tools for privacy-preserving authentication. Unfortunately, known constructions of provably secure blind signatures either rely on non-standard hardness assumptions, or require parameters that grow linearly with the number of concurrently issued signatures, or involve prohibitively inefficient general techniques such as general secure two-party computation.

Recently, Katz, Loss and Rosenberg (ASIACRYPT'21) gave a technique that, for the security parameter n, transforms blind signature schemes secure for $O(\log n)$ concurrent executions of the blind signing protocol into ones that are secure for any $\mathsf{poly}(n)$ concurrent executions.

This transform has two drawbacks that we eliminate in this paper: 1) the communication complexity of the resulting blind signing protocol grows linearly with the number of signing interactions; 2) the resulting schemes inherit a very loose security bound from the underlying scheme and, as a result, require impractical parameter sizes.

In this work, we give an improved transform for obtaining a secure blind signing protocol tolerating any $\mathsf{poly}(n)$ concurrent executions from one that is secure for $O(\log n)$ concurrent executions. While preserving the advantages of the original transform, the communication complexity of our new transform only grows logarithmically with the number of interactions. Under the CDH and RSA assumptions, we improve on this generic transform in terms of concrete efficiency and give (1) a BLS-based blind signature scheme over a standard-sized group where signatures are of size roughly 3 KB and communication per signature is roughly 120 KB; and (2) an Okamoto-Guillou-Quisquater-based blind signature scheme with signatures and communication of roughly 9 KB and 8 KB, respectively.

Keywords. Blind Signatures, Standard Assumptions, Random Oracle Model, Cut-and-Choose.

* Main authors, contributed equally.

This article is a merge of [41] and [9].

1 Introduction

In 1982, David Chaum introduced blind signature schemes in the context of electronic cash [10]. A blind signature scheme is a cryptographic primitive in which a signer can interactively sign a message held by a user. Informally, a blind signature scheme must satisfy two security requirements [28,36]. *Blindness:* the signer should not be able to see what message is being signed. *Unforgeability:* The user should only be able to obtain valid signatures by interacting with the signer. Classical applications of blind signature schemes include e-cash [10,33], anonymous credentials [6,7] and e-voting [23]. Recently, blind signatures have also been used to add privacy features to blockchain-based systems [27]. Despite this variety of promising applications, the current state-of-the art is unsatisfactory. This is because even in the random oracle model, schemes with reasonable efficiency are either based on non-standard assumptions [4,2,15] or have parameters that grow linearly in the number of concurrent signing interactions [36,25,3,30]. The main goal of this work is to construct blind signature schemes from well-established assumptions with concurrent security and practically efficient parameter sizes.

State-of-the-Art. Juels, Luby and Ostrovsky showed that blind signature schemes can be built generically from any secure signature scheme using secure two-party computation [28]. Their construction was only shown secure when signatures were issued *sequentially.* However, typically one aims for the stronger notion of concurrent security. Fischlin [14] achieved this by giving universally composable blind signatures from commitment schemes and UC zero-knowledge proofs; but it is not clear how to instantiate these generic constructions efficiently. While it is tempting to instantiate these schemes with efficient signature schemes in the random oracle model, the security implications of such an instantiation are unclear. This is because such an instantiation would imply the use of the random oracle as a circuit, which constitutes a non-standard use of the random oracle model. We refer to the recent work of [1] which discusses these issues in more detail.

In the standard model, a variety of blind signature schemes have been proposed. These schemes are either inefficient as they rely on complexity leveraging [18] or rely on strong q-type or non-interactive assumptions [32,19,15,20].

Unfortunately, even in the random oracle model, the situation does not improve much. While there are simple constructions [4,2,36,25,26], they either require similar non-standard assumptions as their standard model counterparts [4,2] or support only a very small number of signatures per public key [36,25,26,3].

As a first step to overcome these limitations, Katz, Loss, and Rosenberg (KLR) [30] showed how to use a cut-and-choose technique to boost the security of these blind signature schemes in the random oracle model. Their approach is based on an early work by Pointcheval [35]. The resulting schemes support polynomially many concurrent signature interactions and are based on standard assumptions. However, the communication between the signer and the user still grows linearly with the number of signature interactions, which renders the scheme impractical.

We note that relying on the algebraic or generic group model [16,39] yields better composition and efficiency, as recent works [17,29,40] show. However, these models are best avoided as they are non-standard.

Our Goal. In this work, we advance the state of the art by giving the first blind signature schemes in the random oracle model that do not suffer from any of the above drawbacks. Our main research question can be summarized as follows:

Are there practical and concurrently secure blind signatures from well-established hardness assumptions which support polynomially many signatures?

1.1 Starting Point: The Basic Boosting Transform

We answer this question in the affirmative. We propose several new techniques which reduce the size and communication complexity of blind signatures in the random-oracle model.

Before we explain our techniques, we briefly recall the KLR transform [30], which will serve as our starting point. The KLR transform can be applied to a blind signature scheme BS in which the user sends a single message and which supports a logarithmic number of signing sessions. The transformed scheme CCBS supports polynomially many signing sessions and achieves the same notion of blindness as BS. We briefly recall the main ideas of CCBS before explaining our improved version:

- In the N^{th} signing interaction, the Signer and the User initiate N sessions of the underlying scheme BS. In the i^{th} session, a commitment μ_i of the actual message is signed.
- The User commits to its randomness ρ_i for the i^{th} session using a commitment $\mathsf{com}_i = \mathsf{H}(\rho_i)$, where H is a hash function (modeled as a random oracle). It sends com_i together with its (only) message in the i^{th} session of BS.
- The Signer picks a session $j \in [N]$ uniformly at random and has the User open the randomness to all commitments $\mathsf{com}_i, i \in [N] \setminus \{J\}$.
- If the User cannot open one of these commitments, the Signer aborts. Otherwise, the Signer and User complete the J^{th} session as in BS.

The proof of one-more unforgeability for CCBS is by reduction to the one-more unforgeability of BS. The reduction's goal is to turn a one-more forgery against CCBS into a one-more forgery against BS. To do so, the reduction must answer all signing queries of the User without knowing the secret key sk of the Signer in BS. It is further restricted by the fact that it may invoke the Signing oracle in the underlying security game for BS only logarithmically many times.

To bypass these restrictions, the reduction heavily relies on its capability of observing the inputs to the random oracle and programming it accordingly. Suppose that the the User behaves honestly in Session J, i.e., it uses the randomness in com_J to compute its message in the J^{th} session of BS. Then the reduction can extract the random coins from the commitments and use random oracle

programming to complete this session without knowing sk. If, on the other hand, the User cheats, then the reduction can not use this technique and must ask the Signing oracle in BS for help.

KLR's key observation is that the probability of such a (successful) cheat is at most $1/N$ in the N^{th} signing session. Thus, the expected number of successful cheats in p interactions is at most $\sum_{N \leq p+1} 1/N < \ln(p+1)$. Using the Chernoff bound, one can show that with overwhelming probability, the number of successful cheats is reasonably close to this expectation. Hence, the signing oracle in the underlying OMUF game of BS needs to be invoked only a logarithmic number of times.

Limitations. Although CCBS exponentially increases the security of the underlying blind signature scheme BS, this comes at a steep price in terms of efficiency: the communication in the resulting scheme grows linearly with the number N of issued signatures. This arguably renders CCBS impractical. In addition, the number of times that the reduction from one-more unforgeability of BS requires invoking the underlying signing oracle behaves as $\ln(1/\epsilon)$. Here, ϵ is the advantage of the adversary in breaking one-more unforgeability of CCBS. For small sizes of ϵ (say, 2^{-128}), this leads to impractical parameter sizes for BS. As an example, if CCBS is applied to the Schnorr blind signature scheme, our calculations show that the resulting scheme will require groups with a 12000 bit representation.

1.2 Our Contribution: Improved Boosting Transforms

As our first contribution, we present a new generic transform to boost the security of blind signature schemes fitting the linear function family framework of Hauck, Kiltz and Loss (HKL) [25]. This is based on three insights, as follows. (1) In the N^{th} signing session, the User can derive the random coins for the i^{th} instance via $\rho_i := \mathsf{PRF}(k, i)$, where PRF denotes a *puncturable pseudorandom function* [38][3]. The User can now commit to all its randomness as in CCBS. To open the commitments $\mathsf{com}_i, i \in [N] \setminus \{J\}$, the User provides the punctured key k_J. From this key, the Signer can deterministically recompute all the commitments, save for com_J. (2) We use a randomness homomorphic commitment scheme to construct the μ_i as rerandomizations of one initial commitment μ_0 that is sent to the signer. The rerandomization is also determined by PRF, which implies that k_J also reveals μ_i for $i \neq J$ without revealing μ_J. (3) To compress the N messages from the Signer to the User, we use the homomorphic properties of HKL blind signatures and derive N first messages of the underlying blind signature from $\log N$ randomly chosen ones. These insights allow us to lower the communication complexity of the resulting blind signature scheme from linear to logarithmic in the number N of signing sessions.

Our results have better blindness guarantees than schemes from the KLR transform. A KLR-transformed blind signature scheme has the same blindness as its underlying scheme; for many of the schemes underlying it, only so-called

[3] We instantiate PRF efficiently using random oracles [22].

honest signer blindness was known [25], where the Signer's public key is generated honestly. A much more desirable notion is *malicious signer* blindness, in which the Signer is free to pick his public key adversarially. We show how to achieve this notion using a three step approach. First, we show that the schemes in [25] satisfy a slightly stronger (artificial) notion of blindness without any modification. In this intermediate notion (called *semi-honest signer* blindness), the Signer provides the random coins to generate the public key to the experiment. Next, we show that our improved boosting transform preserves any notion of blindness, including the new one. We then show that by having the signer prove knowledge of the random coins we can transform any scheme that satisfies the intermediate notion into a scheme that satisfies malicious signer blindness.

Practical Schemes from CDH and RSA. Even though our generic transform is an exponential improvement over the state-of-the-art, it still results in schemes that require mega bytes of communication when the number of signatures becomes large (say 2^{30}). On top of this, our generic transform would require large (to the point of being currently impractical) group sizes. To overcome these limitations, we give concrete, 128-bit secure, practical blind signature schemes that satisfy concurrent one-more unforgeability under the CDH and RSA assumptions. We summarize the parameter sizes in Table 1.

Scheme	Nr. of Signatures	$\lvert\mathsf{pk}\rvert$	$\lvert\sigma\rvert$	a	b	Max
$\mathsf{BS}_{\mathsf{RSA}}$ (Section 5)	2^{20}	18.37	7.91	0.02	7.11	7.51
$\mathsf{BS}_{\mathsf{RSA}}$ (Section 5)	2^{30}	18.74	8.66	0.02	7.48	8.08
$\mathsf{PIKA}_{\mathsf{CDH}}$ (Section 4)	2^{20}	3.68	3.16	3.05	26.50	87.50
$\mathsf{PIKA}_{\mathsf{CDH}}$ (Section 4)	2^{30}	3.90	3.16	3.05	26.73	118.20

Table 1. Concrete efficiency of our schemes supporting a given number of signatures and 128 bit security. Here, communication complexity is given as $a \cdot \log(N) + b$, where N is the number of issued signatures so far. Column Max shows the communication complexity for the maximum N. All sizes are in KiloBytes.

Our scheme from CDH is statistically malicious signer blind and builds on Boldyreva's blind version of the BLS signature scheme [4] (which is secure under a one-more version of CDH). We observe that by running our boosting transform for several independent keys in parallel, we can ensure that with overwhelming probability, there will be at least one key for which the User is never able to cheat the Signer. We can leverage this into a reduction that embeds the challenge key pk randomly into one of these keys. Then, with high probability, no cheat ever occurs for pk and the reduction can carry out the simulation without having to ever invoke the signing oracle from the underlying one-more unforgeability experiment. This makes it possible to run the scheme with a standard sized group and assuming no more than hardness of the CDH problem. To reduce the size of our resulting signatures, we can use the aggregatability of the BLS scheme. Overall, our scheme from CDH supports 2^{30} signatures at a size of 3KB and 120KB communication per signature.

Our scheme from RSA does not use parallel repetitions to reduce parameter sizes. Instead, we use the trapdoor provided by the RSA system to improve communication complexity of the generic transform. In this way, the Signer can send a single seed from which the User can deterministically derive several values. The Signer, who needs to know the preimages of these values, can then simply use its trapdoor to learn these preimages and proceed with the remainder of the signing protocol. Overall, our scheme from RSA is statistically semi-honest signer blind and supports 2^{30} signatures at a more balanced size of 9KB per signature and 8KB communication per signature. To upgrade it to malicious signer blindness we can either rely on generic proof systems, or on more efficient ones based on quadratic residuosity [21] or discrete logarithms [8].[4] We emphasize, however, that using proofs from general complexity assumptions may be sufficiently efficient in our context, as the proofs only have to be generated and verified once upon registering the Signer's public key. Therefore, they do not affect the complexity of the signing protocol or the size of our signatures.

2 Preliminaries

The security parameter is $n \in \mathbb{N}$. All algorithms get 1^n implicitly as input. For a finite set S, we write $x \leftarrow\!\!\$\, S$ if x is sampled uniformly at random from S. For a distribution $\mathcal{D}$, we write $x \leftarrow \mathcal{D}$ if x is sampled according to $\mathcal{D}$. For a (probabilistic) algorithm $\mathcal{A}$, we write $y \leftarrow \mathcal{A}(x)$, if y is output from $\mathcal{A}$ on input x with uniformly sampled random coins. We write $y = \mathcal{A}(x;\rho)$ to make the random coins ρ explicit, and $y \in \mathcal{A}(x)$ means that y is a possible output of $\mathcal{A}(x)$. An algorithm is said to be PPT if its running time can be bounded by a polynomial in its input size. We say that a function $f : \mathbb{N} \to \mathbb{R}_+$ is negligible in its input n, if $f \in n^{-\omega(1)}$. For a security game $\mathbf{G}$, we write $\mathbf{G} \Rightarrow b$ to indicate that $\mathbf{G}$ outputs b. We denote the first K natural numbers by $[K] := \{1, \ldots, K\}$, Euler's totient function by φ and the group of units in $\mathbb{Z}_N$ by $\mathbb{Z}_N^*$.

Next, we introduce the cryptographic primitives that we need. We recall the well-known computational assumptions in Supplementary Material Section A. For the definition of puncturable pseudorandom functions, we follow [38].

Definition 1 (Puncturable Pseudorandom Function). *A puncturable pseudorandom function (PPRF) is defined to be a triple of PPT algorithms* $\mathsf{PRF} = (\mathsf{Gen}, \mathsf{Puncture}, \mathsf{Eval})$ *with the following syntax:*

- $\mathsf{Gen}(1^n, 1^{d(n)})$ *takes as input the security parameter* 1^n*, an input length* $1^{d(n)}$ *and outputs a key* k*.*
- $\mathsf{Puncture}(k, X)$ *takes as input a key* k *and a polynomial size set* $\emptyset \neq X \subseteq \mathcal{D} = \{0,1\}^{d(n)}$ *and outputs a punctured key* k_X*.*
- $\mathsf{Eval}(k, x)$ *is deterministic, takes a key* k *and an element* $x \in \mathcal{D}$ *as input and outputs an element* $r \in \mathcal{R} = \{0,1\}^n$*.*

[4] If we rely on these proof systems, our scheme can be proven secure assuming that both the RSA assumption and either of these assumptions hold.

Further, the following security and completeness properties should hold:

- ***Completeness of Puncturing.*** *For any* $d(n) = \mathsf{poly}(n), X \subseteq \{0,1\}^{d(n)}$*, any* $k \in \mathsf{Gen}(1^n, 1^{d(n)})$*, any* $k_X \in \mathsf{Puncture}(k, X)$ *and any* $x' \notin X$ *we have* $\mathsf{Eval}(k, x') = \mathsf{Eval}(k_X, x')$*.*
- ***Pseudorandomness.*** *For any* $d(n) = \mathsf{poly}(n)$ *and any PPT algorithm* $\mathcal{A}$ *the following is negligible:*

$$|\Pr\left[\mathcal{A}(St, k_X, (r_x)_{x\in X}) = 1 \,\middle|\, \begin{array}{l}(X, St) \leftarrow \mathcal{A}(1^n), k \leftarrow \mathsf{Gen}(1^n, 1^{d(n)}),\\ k_X \leftarrow \mathsf{Puncture}(k, X),\\ r_x := \mathsf{Eval}(k, x) \text{ for } x \in X\end{array}\right]$$
$$-\Pr\left[\mathcal{A}(St, k_X, (r_x)_{x\in X}) = 1 \,\middle|\, \begin{array}{l}(X, St) \leftarrow \mathcal{A}(1^n), k \leftarrow \mathsf{Gen}(1^n, 1^{d(n)}),\\ k_X \leftarrow \mathsf{Puncture}(k, X),\\ r_x \leftarrow\!\!\$ \{0,1\}^{r(n)} \text{ for } x \in X\end{array}\right]|.$$

We define a special type of perfectly hiding commitment scheme in which the randomness can be rerandomized publicly. Such commitment schemes can be easily constructed from standard assumptions, see Supplementary Material Section D.

Definition 2 (Randomness Homomorphic Commitment Scheme). *A randomness homomorphic commitment scheme is a tuple of PPT algorithms* $\mathsf{CMT} = (\mathsf{Gen}, \mathsf{Com}, \mathsf{Translate})$ *with the following syntax:*

- $\mathsf{Gen}(1^n)$ *takes as input the security parameter* 1^n *and outputs a commitment key* ck*. We assume that* ck *implicitly defines a message space* $\mathcal{M}_{\mathsf{ck}}$ *and a randomness space* $\mathcal{R}_{\mathsf{ck}}$*. Further, we assume that* $\mathcal{R}_{\mathsf{ck}}$ *is a group with respect to an efficiently computable group operation* $+$*.*
- $\mathsf{Com}(\mathsf{ck}, x; r)$ *takes as input a key* ck*, an element* $x \in \mathcal{M}_{\mathsf{ck}}$*, a randomness* $r \in \mathcal{R}_{\mathsf{ck}}$ *and outputs a commitment* $\mu \in \{0,1\}^*$*.*
- $\mathsf{Translate}(\mathsf{ck}, \mu, r)$ *is deterministic, takes a key* ck*, a commitment* $\mu \in \{0,1\}^*$*, and a randomness* $r \in \mathcal{R}_{\mathsf{ck}}$ *as input and outputs a commitment* μ'*.*

Further, the following security and completeness properties should hold:

- ***Completeness of Translation.*** *For any* $\mathsf{ck} \in \mathsf{Gen}(1^n)$*, and* $x \in \mathcal{M}_{\mathsf{ck}}$ *and any* $r, r' \in \mathcal{R}_{\mathsf{ck}}$*, we have*

$$\mathsf{Translate}(\mathsf{ck}, \mathsf{Com}(\mathsf{ck}, x; r), r') = \mathsf{Com}(\mathsf{ck}, x; r + r').$$

- ***Perfectly Hiding.*** *For any key* ck *and any* $x_0, x_1 \in \mathcal{M}_{\mathsf{ck}}$*, the following distributions are identical:*

$$\{(\mathsf{ck}, x_0, x_1, \mu) \mid r \leftarrow\!\!\$ \mathcal{R}_{\mathsf{ck}}, \mu := \mathsf{Com}(\mathsf{ck}, x_0; r)\} \text{ and}$$
$$\{(\mathsf{ck}, x_0, x_1, \mu) \mid r \leftarrow\!\!\$ \mathcal{R}_{\mathsf{ck}}, \mu := \mathsf{Com}(\mathsf{ck}, x_1; r)\}.$$

- ***Computationally Binding.*** *For any PPT algorithm* $\mathcal{A}$*, the following is negligible:*

$$\Pr\left[\mathsf{Com}(\mathsf{ck}, x_0; r_0) = \mathsf{Com}(\mathsf{ck}, x_1; r_1) \wedge x_0 \neq x_1 \,\middle|\, \begin{array}{l}\mathsf{ck} \leftarrow \mathsf{Gen}(1^n),\\ (x_0, r_0, x_1, r_1) \leftarrow \mathcal{A}(\mathsf{ck})\end{array}\right].$$

Next, we define the primitive of interest, namely blind signature scheme.

Definition 3 (Blind Signature Scheme). *A blind signature scheme* $\mathsf{BS} = (\mathsf{Gen}, \mathsf{S}, \mathsf{U}, \mathsf{Ver})$ *is a quadruple of PPT algorithms, where*

- $\mathsf{Gen}(1^n)$ *is a PPT algorithm that outputs a pair of keys* $(\mathsf{pk}, \mathsf{sk})$*. We assume that the public key* pk *defines a message space* $\mathcal{M} = \mathcal{M}_{\mathsf{pk}}$ *implicitly.*
- S *and* U *are interactive algorithms, where* S *takes as input a key pair* $(\mathsf{pk}, \mathsf{sk})$ *and* U *takes as input a key* pk *and a message* $\mathsf{m} \in \mathcal{M}$*. After the execution,* U *returns a signature* σ *and we write* $(\perp, \sigma) \leftarrow \langle \mathsf{S}(\mathsf{sk}), \mathsf{U}(\mathsf{pk}, \mathsf{m}) \rangle$*.*
- $\mathsf{Ver}(\mathsf{pk}, \mathsf{m}, \sigma)$ *is deterministic and takes as input public key, message* $\mathsf{m} \in \mathcal{M}$ *and a signature* σ *and returns* $b \in \{0, 1\}$*.*

We say that BS *is complete if for all* $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{Gen}(1^n)$ *and all* $\mathsf{m} \in \mathcal{M}_{\mathsf{pk}}$ *it holds that*

$$\Pr\left[\mathsf{Ver}(\mathsf{pk}, \mathsf{m}, \sigma) = 1 \mid (\perp, \sigma) \leftarrow \langle \mathsf{S}(\mathsf{sk}), \mathsf{U}(\mathsf{pk}, \mathsf{m}) \rangle\right] = 1.$$

Definition 4 (One-More Unforgeability). *Let* $\mathsf{BS} = (\mathsf{Gen}, \mathsf{S}, \mathsf{U}, \mathsf{Ver})$ *be a blind signature scheme and* $\ell : \mathbb{N} \to \mathbb{N}$*. For an adversary* $\mathcal{A}$*, we consider the following game* ℓ-$\mathbf{OMUF}^{\mathcal{A}}_{\mathsf{BS}}(n)$*:*

1. *Sample keys* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^n)$*.*
2. *Let* O *be an interactive oracle simulating* $\mathsf{S}(\mathsf{sk})$*. Run*

$$((\mathsf{m}_1, \sigma_1), \ldots, (\mathsf{m}_k, \sigma_k)) \leftarrow \mathcal{A}^{O}(\mathsf{pk}),$$

   where $\mathcal{A}$ *can query* O *in an arbitrarily interleaved way and complete at most* $\ell = \ell(n)$ *of the interactions with* O*.*
3. *Output* 1 *if and only if all* $\mathsf{m}_i, i \in [k]$ *are distinct,* $\mathcal{A}$ *completed at most* $k - 1$ *interactions with* O *and for each* $i \in [k]$ *it holds that* $\mathsf{Ver}(\mathsf{pk}, \mathsf{m}_i, \sigma_i) = 1$*.*

We say that BS *is* ℓ*-one-more unforgeable (*ℓ-$\mathsf{OMUF}*), if for every PPT algorithm* $\mathcal{A}$ *the following advantage is negligible:*

$$\Pr\left[\ell\text{-}\mathbf{OMUF}^{\mathcal{A}}_{\mathsf{BS}}(n) \Rightarrow 1\right].$$

Further, we say that BS *is one-more unforgeable (*OMUF*), if it is* ℓ-OMUF *for all polynomial* ℓ*.*

We note that from a practical perspective, it is sufficient to focus on ℓ-OMUF for some large but a priori bounded ℓ (e.g. $\ell = 2^{30}$), while full OMUF is more of theoretical interest.

Definition 5 (Blindness). *Consider a blind signature scheme* $\mathsf{BS} = (\mathsf{Gen}, \mathsf{S}, \mathsf{U}, \mathsf{Ver})$*. For an adversary* $\mathcal{A}$ *and bit* $b \in \{0, 1\}$*, consider the following game* $\mathbf{BLIND}^{\mathcal{A}}_{b,\mathsf{BS}}(n)$*:*

1. *Sample* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^n)$ *and run* $(\mathsf{m}_0, \mathsf{m}_1, St) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{sk})$*.*

2. *Let O_0 be an interactive oracle simulating $\mathsf{U}(\mathsf{pk}, \mathsf{m}_b)$ and O_1 be an interactive oracle simulating $\mathsf{U}(\mathsf{pk}, \mathsf{m}_{1-b})$. Run $\mathcal{A}$ on input St with arbitrary interleaved one-time access to each of these oracles, i.e. $St' \leftarrow \mathcal{A}^{O_0, O_1}(St)$.*
3. *Let σ_b, σ_{1-b} be the local outputs of O_0, O_1, respectively. If $\sigma_0 = \perp$ or $\sigma_1 = \perp$, then run $b' \leftarrow \mathcal{A}(St', \perp, \perp)$. Else, obtain a bit b' from $\mathcal{A}$ on input σ_0, σ_1, i.e. run $b' \leftarrow \mathcal{A}(St', \sigma_0, \sigma_1)$.*
4. *Output b'.*

We say that BS *satisfies honest signer blindness, if for every PPT algorithm $\mathcal{A}$ the following advantage is negligible:*

$$\left| \Pr\left[\mathbf{BLIND}^{\mathcal{A}}_{0,\mathsf{BS}}(n) \Rightarrow 1\right] - \Pr\left[\mathbf{BLIND}^{\mathcal{A}}_{1,\mathsf{BS}}(n) \Rightarrow 1\right] \right|.$$

We also consider semi-honest and malicious signer blindness, where we modify the game in the following way:

- *For semi-honest signer blindness,* $(\mathsf{pk}, \mathsf{sk})$ *is not sampled by the game, but $\mathcal{A}$ outputs random coins ρ in addition to $\mathsf{m}_0, \mathsf{m}_1$. Then, the game defines* $(\mathsf{pk}, \mathsf{sk})$ *via* $(\mathsf{pk}, \mathsf{sk}) := \mathsf{Gen}(1^n; \rho)$.
- *For malicious signer blindness,* $(\mathsf{pk}, \mathsf{sk})$ *is not sampled by the game, but $\mathcal{A}$ outputs* pk *in addition to $\mathsf{m}_0, \mathsf{m}_1$.*

Semi-honest signer blindness is a non-standard notion and lies inbetween honest and malicious signer blindness. We claim that any semi-honest signer blind scheme can be transformed into a malicious signer blind scheme while preserving one-more unforgeability. The high-level idea is to append a non-interactive zero-knowledge proof-of-knowledge to the public key. This proof shows that the signer knows corresponding random coins that generate the key. The rest of the scheme does not change, and thus the transformation is very efficient. For details, see Supplementary Material Section B.

We will now introduce linear function families, following [25].

Definition 6 (Linear Function Family). *A linear function family* LF *is a given by a tuple of algorithms* $\mathsf{LF} = (\mathsf{PGen}, \mathsf{F}, \Psi)$ *with the following properties:*

- $\mathsf{PGen}(1^n)$ *returns system parameters* par *which define abelian groups $\mathcal{S}, \mathcal{D}, \mathcal{R}$ with $|\mathcal{S}|, |\mathcal{R}| \geq 2^n$ 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 $(s + s') \cdot x = s \cdot x + s' \cdot x$.*
- $\mathsf{F}_{\mathsf{par}}(x)$ *is deterministic, takes as input an element $x \in \mathcal{D}$, and returns an element in $y \in \mathcal{R}$. We require that:*
  - *For all $s \in \mathcal{S}, x, y \in \mathcal{D}$, $\mathsf{F}_{\mathsf{par}}(s \cdot x + y) = s \cdot \mathsf{F}_{\mathsf{par}}(x) + \mathsf{F}_{\mathsf{par}}(y)$.*
  - $\mathsf{F}_{\mathsf{par}}$ *has a pseudo torsion-free element in the kernel, i.e. there exists $z^* \in \mathcal{D}$ such that $\mathsf{F}_{\mathsf{par}}(z^*) = 0$ and for all distinct $s, s' \in \mathcal{S}, s \cdot z^* \neq s' \cdot z^*$.*
  - $\mathsf{F}_{\mathsf{par}}$ *is smooth, i.e. if $x \leftarrow \mathcal{D}$ is sampled uniformly, $\mathsf{F}_{\mathsf{par}}(x)$ has uniform distribution in $\mathcal{R}$.*

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

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

Intuitively, the function Ψ_{par} *corrects for the fact that the group operation in* $\mathcal{S}$ *may not distribute over* $\mathcal{R}$*. When it is clear from the context, we will omit the subscript* par*.*

As in [30], we define preimage resistance for a linear function family. For the related notion of collision resistence, we refer to Supplementary Material Section C.

Definition 7 (Preimage Resistance). *A linear function family* LF *is preimage resistant if for any adversary* $\mathcal{A}$*, the following advantage is negligible:*

$$\Pr\left[\mathsf{F}(x) = \mathsf{F}(x') | x \leftarrow \mathcal{D}, x' \leftarrow \mathcal{A}(\mathsf{par}, \mathsf{F}(x))\right].$$

3 An Improved Boosting Transform

Hauck, Kiltz, and Loss [25] introduced a generic construction of a three-move blind signature scheme $\mathsf{BS}[\mathsf{LF}]$ from any linear function family LF and a hash function H modeled as a random-oracle. The main result of [25] is that the linear blind signature scheme $\mathsf{BS}[\mathsf{LF}]$ is ℓ-one-more unforgeable for $\ell = \mathcal{O}(\log n)$. Building on that, Katz, Loss, and Rosenberg [30] presented a boosting transform $\mathsf{CCBS}[\mathsf{LF}]$ that turns this logarithmic security into polynomial security. In this section, we introduce an improved boosting transform $\mathsf{CCCBS}[\mathsf{LF}]$ that eliminates the drawback of linearly growing communication complexity.

3.1 Overview

We recall the main idea of the boosting transform [30] that turns a linear blind signature scheme $\mathsf{BS}[\mathsf{LF}]$ into a boosted blind signature scheme $\mathsf{CCBS}[\mathsf{LF}]$.

In the scheme $\mathsf{CCBS}[\mathsf{LF}]$, at the onset of the N^{th} interaction, the signer sends the current value of the counter N to the user. Then, user and signer proceed as follows.

1. The user chooses N random strings $\mathsf{ur}_j, j \in [N]$ and N random strings $\varphi_j, j \in [N]$. It prepares N commitments $\mu_j = \mathsf{H}(\mathsf{m}, \varphi_j)$, where H is a random oracle and m is the message to be signed. It also prepares commitments $\mathsf{com}_j = \mathsf{H}(\mathsf{ur}_j, \mu_j)$. Then it sends the commitments com_j to the signer.
2. The user and the signer run N independent sessions of the underlying linear blind signature scheme $\mathsf{BS}[\mathsf{LF}]$, where the user inputs μ_j, ur_j in the j^{th} session. Recall that the scheme $\mathsf{BS}[\mathsf{LF}]$ contains three messages R, c, s.
3. Before the signer sends the last message s_j of the underlying scheme, it chooses a cut-and-choose index $J \in [N]$ at random and asks the user to open all commitments com_j with $j \neq J$.

4. Once the signer knows the values μ_j and randomness ur_j, it runs the user algorithm U to check if the user behaved honestly so far, at least for the sessions $j \neq J$. If there is some session for which this check fails, the signer aborts.
5. The signer sends only s_J to the user. That is, signer and user only complete the J^{th} session. The final signature consists of a signature on μ_J from the underlying scheme $\mathsf{BS}[\mathsf{LF}]$ as well as the randomness φ_J which binds m to μ_J.

We highlight that the communication now grows linearly with the number of issued signatures.

a) In the second message, the user sends N commitments com_j.
b) In the third message, the signer sends N commitments R_j.
c) In the fourth message, the user sends N challenges c_j.
d) In the sixth message, the user opens each of the N commitments com_j.

Our goal is to eliminate these linear dependencies on N and improve them by an at most logarithmic dependency.

First, we eliminate the linear dependency a) by replacing the commitments $\mathsf{com}_j = \mathsf{H}(\mathsf{ur}_j, \mu_j)$ by a single commitment com_r, which commits to (salted) hashes of all ur_j, μ_j at once. By sending all ur_j for $j \neq J$ and the hash of ur_J, the user can still open this commitment without revealing ur_J.

Next, we focus on d). Here, we let the user generate the randomness $(\mathsf{ur}_j, \varphi_j)$ used for each session using the puncturable pseudorandom function PRF. We replace the unstructured commitment with a randomness homomorphic commitment scheme. This allows us to let the user derive the commitments μ_i as rerandomizations $\mathsf{Com}(\mathsf{m}, \varphi_0 + \varphi_j)$ of one single commitment $\mu_0 = \mathsf{Com}(\mathsf{m}, \varphi_0)$ with randomness φ_j. The user sends commitment μ_0 together with com_r. Now, the user can open the commitment com_r by sending only a punctured key k_J. Intuitively, this preserves blindness, as the punctured key does not reveal anything about the randomness ur_J, φ_J. Using similar tricks, we eliminate c).

To tackle b), we compute the N values R_i of the underlying linear scheme $\mathsf{BS}[\mathsf{LF}]$ as subset sums of a logarithmic number of such values. Then, only these basis values have to be sent.

We end up with a scheme with logarithmic communication complexity, for which the ideas that underlie the original boosting transform still apply.

3.2 Blind Signatures from Linear Function Families

We briefly recall the blind signature scheme $\mathsf{BS}[\mathsf{LF}]$ from a linear function family LF. For more details, we refer the reader to [25] or Supplementary Material Section C. For key generation of the blind signature scheme $\mathsf{BS}[\mathsf{LF}]$, parameters $\mathsf{par} \leftarrow \mathsf{LF.PGen}(1^n)$ are generated. Then, a secret key and public key are sampled via $\mathsf{sk} \leftarrow\!\!\$\, \mathcal{D}$ and $\mathsf{pk} := \mathsf{F}(\mathsf{sk})$, assuming pk implicitly contains par. We present the signature issuing protocol formally in Figure 1. Signatures $\sigma = (c', s')$ for a message m are verified by checking if $c' = \mathsf{H}(\mathsf{m}, \mathsf{F}(s') - c' \cdot \mathsf{pk})$ holds.

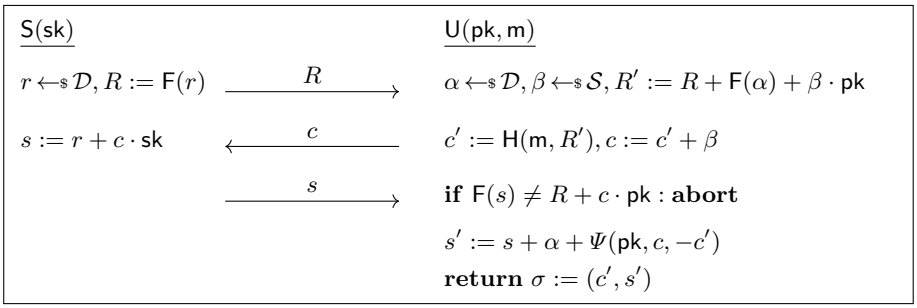

Fig. 1. The signature issuing protocol of the linear blind signature scheme $\mathsf{BS}[\mathsf{LF}]$ for a linear function family LF and a random oracle $\mathsf{H} : \{0,1\}^* \to \mathcal{S}$ [25].

3.3 Construction

In this section, we define our Compact Cut-and-Choose blind signature scheme for a linear function family LF, abbreviated as $\mathsf{CCCBS}[\mathsf{LF}]$. To this end, let $\mathsf{LF} = (\mathsf{PGen}, \mathsf{F}, \Psi)$ be a linear function family, CMT be a randomness homomorphic commitment scheme, and PRF be a puncturable pseudorandom function. For efficient instantiations of CMT and PRF, see Supplementary Material Section D and Supplementary Material Section E. Further, let $\mathsf{H} : \{0,1\}^* \to \mathcal{S}, \mathsf{H}_\mathsf{r} : \{0,1\}^* \to \{0,1\}^n, \mathsf{H}_x : \{0,1\}^* \to \mathcal{D} \times \mathcal{S} \times \mathcal{R}_\mathsf{ck} \times \{0,1\}^{n_\mathsf{PRF}}, \mathsf{H}_c : \{0,1\}^* \to \{0,1\}^n$ be random oracles, where $n_\mathsf{PRF} = \Theta(n)$ is a security parameter used for the pseudorandom function.

Key Generation. Algorithm $\mathsf{CCCBS}[\mathsf{LF}].\mathsf{Gen}(1^n)$ is as follows:

1. Sample $\mathsf{ck} \leftarrow \mathsf{CMT}.\mathsf{Gen}(1^n)$ and $\mathsf{par} \leftarrow \mathsf{LF}.\mathsf{PGen}(1^n)$.
2. Sample $\mathsf{sk}' \leftarrow\!\!\$\ \mathcal{D}$, and let $\mathsf{sk} := \mathsf{sk}'$, $\mathsf{pk} = (\mathsf{par}, \mathsf{ck}, \mathsf{pk}' := \mathsf{F}(\mathsf{sk}'))$.
3. Return the public key pk and the secret key sk.

Signature Issuing. The signer and user algorithms S, U are given in Figures 2 and 3, where the S keeps a state (N, ctr) which is initialized as $N := 2 = 2^2 - 2, \mathsf{ctr} := 0$. In each interaction, S atomically increments ctr and, if $\mathsf{ctr} = N$, sets $N := 2N + 2, \mathsf{ctr} := 0$.

Verification. Algorithm $\mathsf{CCCBS}[\mathsf{LF}].\mathsf{Ver}(\mathsf{pk}, \mathsf{m}, \sigma = (c, s, \varphi))$ returns the output of $\mathsf{BS}[\mathsf{LF}].\mathsf{Ver}(\mathsf{pk}', \mathsf{Com}(\mathsf{ck}, \mathsf{m}; \varphi), (c, s))$.

3.4 Security Analysis

Completeness of $\mathsf{CCCBS}[\mathsf{LF}]$ follows by inspection. We show blindness and one-more unforgeability.

$\mathsf{Check}(\mathsf{pk}, N, \mu_0, \mathsf{com}_\mathsf{r}, \{R_i\}_i, \mathsf{com}_c, J, k_J, c_J, h_J)$

1 : **for** $j \in [N] \setminus \{J\}$:
2 : $\quad \mathsf{prer}_j := \mathsf{PRF.Eval}(k_J, j),\ \mathsf{r}_j := \mathsf{H}_x(\mathsf{prer}_j)$
3 : $\quad$ **parse** $\mathsf{r}_j = (\alpha_j, \beta_j, \varphi_j, \gamma_j) \in \mathcal{D} \times \mathcal{S} \times \mathcal{R}_\mathsf{ck} \times \{0,1\}^{n_\mathsf{PRF}}$
4 : $\quad \tilde{R}_j := \sum_{i \in S_j} R_j, \quad \mu_j := \mathsf{Translate}(\mathsf{ck}, \mu_0, \varphi_j)$
5 : $\quad c_j := \mathsf{H}(\mu_j, \tilde{R}_j + \mathsf{F}(\alpha_j) + \beta_j \cdot \mathsf{pk}') + \beta_j$
6 : **if** $\mathsf{com}_\mathsf{r} \neq \mathsf{H}_\mathsf{r}(\mathsf{H}_\mathsf{r}(\mathsf{r}_1), \ldots, \mathsf{H}_\mathsf{r}(\mathsf{r}_{J-1}), h_J, \mathsf{H}_\mathsf{r}(\mathsf{r}_{J+1}), \ldots, \mathsf{H}_\mathsf{r}(\mathsf{r}_N))$: **return** 0
7 : **if** $\mathsf{com}_c \neq \mathsf{H}_c(c_1, \ldots, c_N)$: **return** 0
8 : **return** 1

Fig. 2. The algorithm Check used in the issuing protocol of CCCBS[LF], where $\mathsf{H} : \{0,1\}^* \to \mathcal{S}, \mathsf{H}_\mathsf{r}, \mathsf{H}_c : \{0,1\}^* \to \{0,1\}^n$, and $\mathsf{H}_x : \{0,1\}^* \to \mathcal{D} \times \mathcal{S} \times \mathcal{R}_\mathsf{ck} \times \{0,1\}^{n_\mathsf{PRF}}$ are random oracles. The set S_j is defined as $\{i \in [l] : i^{th}\text{-bit of } j \text{ is } 1\}$.

Theorem 1. *Let* PRF *be a puncturable pseudorandom function,* LF *be a linear function family, and* CMT *be a randomness homomorphic commitment scheme. Let* $\mathsf{H}_\mathsf{r} : \{0,1\}^* \to \{0,1\}^n, \mathsf{H}_x : \{0,1\}^* \to \mathcal{D} \times \mathcal{S} \times \mathcal{R}_\mathsf{ck} \times \{0,1\}^{n_\mathsf{PRF}}$ *be random oracles. If* BS[LF] *satisfies honest, semi-honest, or malicious signer blindness, then* CCCBS[LF] *satisfies honest, semi-honest, or malicious signer blindness, respectively.*

Concretely, for any adversary that uses N^L *and* N^R *as the counters in its executions with the user, runs in time* t*, has advantage* ϵ *in the blindness game and makes at most* $Q_{\mathsf{H}_x}, Q_{\mathsf{H}_\mathsf{r}}$ *queries to* $\mathsf{H}_x, \mathsf{H}_\mathsf{r}$ *respectively, there exists an adversary against blindness of* BS[LF] *running in time* t *with advantage* $\epsilon_{\mathsf{BS[LF]}}$ *such that*

$$\epsilon \leq N^L N^R \left(\frac{4(Q_{\mathsf{H}_x} + Q_{\mathsf{H}_\mathsf{r}})}{2^{n_\mathsf{PRF}}} + 4\epsilon_\mathsf{PRF} + \epsilon_{\mathsf{BS[LF]}} \right),$$

where ϵ_PRF *is the advantage of an adversary against the security of* PRF *with input length* $\max\{\log(N^L), \log(N^R)\}$ *puncturing at one point.*

We give a intuition of the proof and postpone details to Supplementary Material Section G.1. The strategy is to apply a sequence of changes to the user oracles, such that final game is independent of bit b. In a first step, we guess the cut-and-choose index J. Then, we compute the commitment μ_J directly instead of deriving it from the commitment μ_0. Next, we use the security of PRF and generate r_J for session J at random instead of using the key k. Now, we observe that the randomness φ_0 is hidden in the final signature, and we can switch μ_0 to a commitment of a random message. Finally, we see that the only dependency on the message is in session J and we can reduce from the blindness of BS[LF].

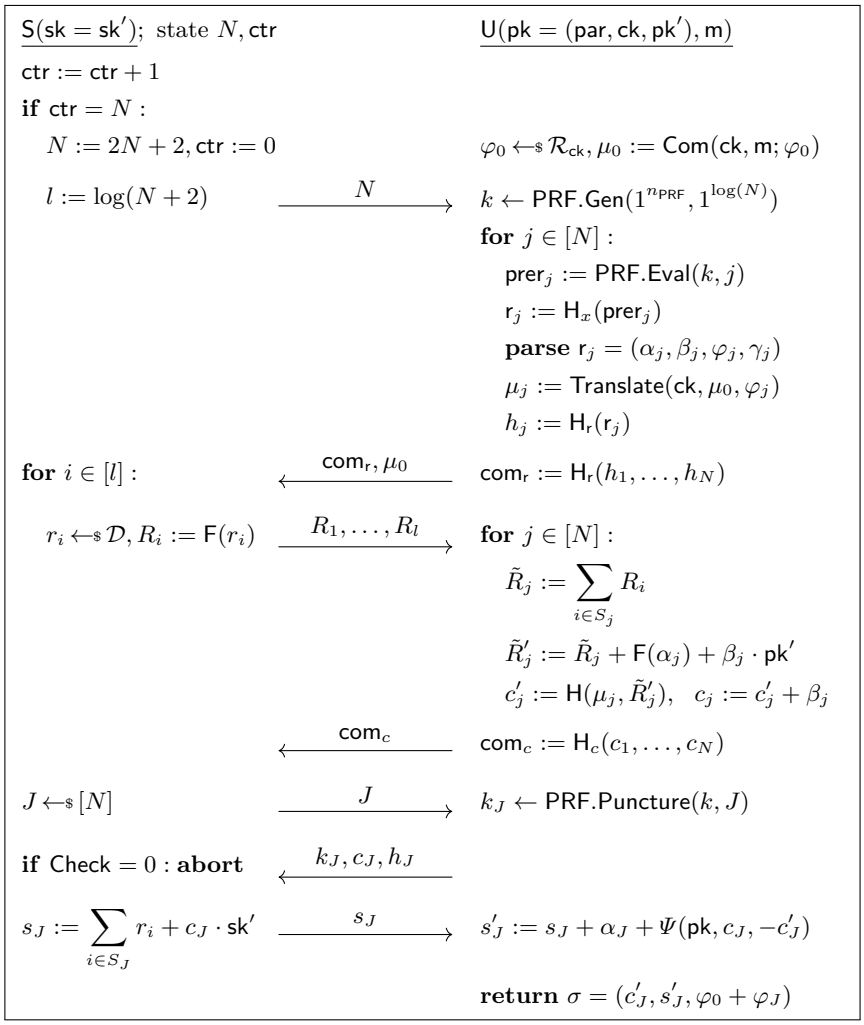

Fig. 3. The signature issuing protocol of the blind signature scheme CCCBS[LF], where $\mathsf{H} : \{0,1\}^* \to \mathcal{S}, \mathsf{H}_\mathsf{r}, \mathsf{H}_c : \{0,1\}^* \to \{0,1\}^n, \mathsf{H}_x : \{0,1\}^* \to \mathcal{D} \times \mathcal{S} \times \mathcal{R}_\mathsf{ck} \times \{0,1\}^{n_\mathsf{PRF}}$ are random oracles. The algorithm Check is defined in Figure 2. The set S_j is defined as $\{i \in [l] : i^{th}\text{-bit of } j \text{ is } 1\}$. The states ctr and N are incremented atomically.

Theorem 2. *Let* PRF *be a puncturable pseudorandom function,* LF *be a linear function family, and* CMT *be a randomness homomorphic commitment scheme. Let* $\mathsf{H} : \{0,1\}^* \to \mathcal{S}, \mathsf{H}_\mathsf{r}, \mathsf{H}_c : \{0,1\}^* \to \{0,1\}^n$ *be random oracles. If* BS[LF] *satisfies ℓ-one-more unforgeability for any $\ell = O(\log(n))$, then* CCCBS[LF] *satisfies ℓ-one-more unforgeability for any $\ell = \mathsf{poly}(n)$.*

Concretely, suppose there exists an adversary with advantage ϵ against the ℓ-one-more unforgeability of $\mathsf{CCCBS}[\mathsf{LF}]$*, that runs in time t, starts at most p interactions with his signer oracle, and makes at most $Q_{\mathsf{H}}, Q_{\mathsf{H}_\mathsf{r}}, Q_{\mathsf{H}_c}$ queries to $\mathsf{H}, \mathsf{H}_\mathsf{r}, \mathsf{H}_c$ respectively. Then, there exists an adversary against the λ-one-more unforegability* $\mathsf{BS}[\mathsf{LF}]$*, where $\lambda = 3\lceil \log p \rceil + \log(2/\epsilon)$, that runs in time t, starts at most p interactions with his signer oracle, makes at most Q_{H} queries to H, and has advantage $\epsilon_{\mathsf{BS}[\mathsf{LF}]}$, such that*

$$\epsilon \leq 2\left(\epsilon_{\mathsf{BS}[\mathsf{LF}]} + p \cdot \epsilon_{\mathsf{LF}} + \epsilon_{\mathsf{CMT}} + \frac{Q_{\mathsf{H}_\mathsf{r}}^2 + Q_{\mathsf{H}_c}^2 + pQ_{\mathsf{H}_\mathsf{r}} + pQ_{\mathsf{H}_c}}{2^n} + \frac{p^2(p^2 + Q_{\mathsf{H}})}{|\mathcal{R}|}\right),$$

where ϵ_{LF} is the advantage of an adversary with running time t against the preimage resistance of LF *and ϵ_{CMT} is the advantage of an adversary with running time t against the binding property of* CMT*.*

The proof is very similar to the proof for the original boosting transform [30]. Thus, we postpone it to Supplementary Material Section G.2.

Remark 1. As an asymptotic result, we are satisfied with our improved boosting transform with logarithmic communication complexity. However, similar to the original boosting transform, we rely on the very loose security bound of the underlying linear blind signature scheme $\mathsf{BS}[\mathsf{LF}]$. For concrete efficiency, this is prohibitive, as we require that $\mathsf{BS}[\mathsf{LF}]$ supports a non-trivial number λ of signatures. Also, the logarithmic term of the communication complexity depends on computational assumptions. Thus, the loose bound will also have a negative impact on communication complexity.

To highlight this, we computed the parameter sizes for the instantiations of the boosting transform based on the discrete logarithm problem. Our calculations show that in order to support 2^{30} signatures, the scheme requires a 12035 bit group. It is apparent that this group size is impractical, and no standardized elliptic curve groups of this size exist. We remark that Katz et al. [30] also provide a parameter estimate, but this holds only for a very specific choice of signing queries, random oracle queries and advantage. For a detailed explanation of our calculations, see Supplementary Material Section J.

In the following, we will see how to augment the ideas of this section to construct schemes which eliminate aforementioned drawbacks and come with practical concrete parameters.

4 A Concrete Scheme based on CDH

Here, we construct a concrete blind signature scheme $\mathsf{PIKA}_{\mathsf{CDH}}$ based on the CDH assumption. While the construction in the previous section was generic, we aim for a scheme with concrete efficiency in this section.

4.1 Overview

As discussed in Remark 1, our improved boosting transform inherits the loose security bound of the underlying linear blind signature scheme. To see how we can circumvent this, let us first recall the reduction idea of the boosting transform. The main challenge is that the underlying scheme $\mathsf{BS}[\mathsf{LF}]$ allows for a logarithmic number of signing interactions, while the reduction has to simulate an arbitrary polynomial number of signing interactions for the adversary. This is solved as follows. First, note that whenever the adversary honestly commits to ur_j, μ_j, the reduction can extract these values from the commitments com_r by observing the random oracle queries. Then, an important property of linear blind signature schemes comes into play: If one knows the randomness and the message that is input into the user algorithm $\mathsf{BS}[\mathsf{LF}].\mathsf{U}$ and controls the random oracle, one can simulate the signer algorithm without knowing the secret key. Thus, the reduction only needs to access the signer oracle of $\mathsf{BS}[\mathsf{LF}]$ if the adversary *cheats* (i.e., it malforms the commitment for the J^{th} session in the first step and is not caught). Fortunately, the probability of such a (successful) cheat is at most $1/N$ in the N^{th} signing session. Thus, the expected number of successful cheats in p interactions is at most logarithmic in p. Using the Chernoff bound, one can show that with overwhelming probability, the number of successful cheats is reasonably close to this expectation.

We observe that by letting the cut-and-choose parameter grow slightly faster than before and scaling appropriately, the expected number of successful cheats can be bounded to be less than 1. Unfortunately, we can not just use the Chernoff bound, if we want to argue that this also holds with overwhelming probability. We can, however, use the Chernoff bound to show that exceeding a single cheat happens with some constant probability less than 1. Then, we play our next card, which is parallel repetition. Namely, we run K independent instances of our scheme so far, where each instance is relative to a separate key pair. We show that with high probability, in one randomly chosen instance, there is *no cheat at all.* Using this observation, we can give a reduction from the key-only security of the underlying blind signature scheme to finish our proof.

We do not apply this overall strategy to a linear blind signature scheme, but instead to the BLS blind signature scheme [4]. We notice that the approach also works for this scheme and observe additional benefits: First, the BLS scheme allows to aggregate signatures. Hence, it is easy to merge the resulting signatures from the K instances for a significant efficiency improvement. Second, the scheme has two rounds and thus the logarithmic term in the communication complexity is independent of computational assumptions (cf. Remark 1). We emphasize that the original BLS blind signature scheme is secure under a one-more variant of the CDH assumption. Fortunately, we only need key-only security here, which is implied by CDH. Also, the concrete security loss of our scheme is as for the standard BLS digital signature scheme [5], which means that it can be used over the same groups as BLS.

Finally, we introduce further minor optimizations such as making the signer commit to its cut-and-choose indices in its message. In this way, the reduction in

the blindness proof can extract these indices rather than guessing them. This leads to more efficient statistical security parameters [5].

4.2 Construction

Let $\mathsf{PGGen}(1^n)$ be a bilinear group generation algorithm that outputs a cyclic group $\mathbb{G}$ of prime order p with generator g, and a pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ into some target group $\mathbb{G}_T$. We assume that these system parameters are known to all algorithms. Note that their correctness can be verified efficiently. Our scheme makes use of a randomness homomorphic commitment scheme CMT with randomness space $\mathcal{R}_{\mathsf{ck}}$ and a puncturable pseudorandom function PRF. We can instantiate PRF using random oracles (cf. Supplementary Material Section E) and CMT tightly based on the DLOG assumption (cf. Supplementary Material Section D.2). We also need random oracles $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_p, \mathsf{H}' : \{0,1\}^* \to \{0,1\}^n$ and $\mathsf{H}_\mathsf{r}, \mathsf{H}_c : \{0,1\}^* \to \{0,1\}^n, \mathsf{H}_x : \{0,1\}^* \to \mathbb{Z}_p \times \mathcal{R}_{\mathsf{ck}} \times \{0,1\}^{n_{\mathsf{PRF}}}$, where n_{PRF} is a security parameter used for PRF.

Our scheme makes use of a parameter $K \in \mathbb{N}$, which defines how many instances of the underlying boosting transform are executed in parallel, and a function $f : \mathbb{N} \to \mathbb{N}$, which determines how fast the cut-and-choose parameter N grows. In Section 4.4, we will discuss how to set these parameters efficiently.

Key Generation. To generate keys algorithm $\mathsf{PIKA}_{\mathsf{CDH}}.\mathsf{Gen}(1^n)$ does the following:

1. For each instance $i \in [K]$, sample $\mathsf{sk}_i \leftarrow\!\!\$\, \mathbb{Z}_p$ and set $\mathsf{pk}_i := g^{\mathsf{sk}_i}$.
2. Sample a commitment key $\mathsf{ck} \leftarrow \mathsf{CMT}.\mathsf{Gen}(1^n)$.
3. Return public key $\mathsf{pk} := (\mathsf{pk}_1, \ldots, \mathsf{pk}_K, \mathsf{ck})$ and secret key $\mathsf{sk} := (\mathsf{sk}_1, \ldots, \mathsf{sk}_K)$.

Signature Issuing. The algorithms S, U and their interaction are formally given in Figures 4 and 5. Here, S keeps a state ctr, which is inititalized as $\mathsf{ctr} := 1$ and incremented in every interaction.

Verification. The resulting signature $\sigma = (\bar{\sigma}, \varphi_1, \ldots, \varphi_K)$ for a message m is verified by algorithm $\mathsf{PIKA}_{\mathsf{CDH}}.\mathsf{Ver}(\mathsf{pk}, \mathsf{m}, \sigma)$ as follows:

1. For each instance $i \in [K]$, compute the commitment $\mu_i := \mathsf{Com}(\mathsf{ck}, \mathsf{m}; \varphi_i)$.
2. Return 1 if and only if

$$e\left(\bar{\sigma}, g\right) = \prod_{i=1}^{K} e\left(\mathsf{H}(\mathsf{pk}_i, \mu_i), \mathsf{pk}_i\right).$$

[5] Note that without this optimization, the security loss would be exponential in K.

$\mathsf{Check}(\mathsf{pk}, N, \mu_0, \mathsf{com}_\mathsf{r}, \mathsf{com}_c, \mathsf{seed}_\mathbf{J}, k_\mathbf{J}, \{c_{i,\mathbf{J}_i}\}_i, \{\eta_i\}_i)$

$\mathbf{J} = (\mathsf{H}'(\mathsf{seed}_\mathbf{J}, 1), \ldots, \mathsf{H}'(\mathsf{seed}_\mathbf{J}, K)) \in [N]^K$
for $i \in [K]$:
  for $j \in [N] \setminus \{\mathbf{J}_i\}$:
    $\mathsf{prer}_{i,j} := \mathsf{PRF.Eval}(k_\mathbf{J}, (i,j)),\ \mathsf{r}_{i,j} := \mathsf{H}_x(\mathsf{prer}_{i,j})$
    parse $\mathsf{r}_{i,j} = (\alpha_{i,j}, \varphi_{i,j}, \gamma_{i,j}) \in \mathbb{Z}_p \times \mathcal{R}_\mathsf{ck} \times \{0,1\}^n$
    $\mu_{i,j} := \mathsf{Translate}(\mathsf{ck}, \mu_0, \varphi_{i,j})$
    $c_{i,j} := \mathsf{H}(\mathsf{pk}_i, \mu_{i,j}) \cdot g^{\alpha_{i,j}}$
  $\mathsf{com}_{\mathsf{r},i} := \mathsf{H}_\mathsf{r}(\mathsf{H}_\mathsf{r}(\mathsf{r}_{i,1}), \ldots, \mathsf{H}_\mathsf{r}(\mathsf{r}_{i,\mathbf{J}_i-1}), \eta_i, \mathsf{H}_\mathsf{r}(\mathsf{r}_{i,\mathbf{J}_i+1}), \ldots, \mathsf{H}_\mathsf{r}(\mathsf{r}_{i,N}))$
if $\mathsf{com}_\mathsf{r} \neq \mathsf{H}_\mathsf{r}(\mathsf{com}_{\mathsf{r},1}, \ldots, \mathsf{com}_{\mathsf{r},K})$: **return** 0
if $\mathsf{com}_c \neq \mathsf{H}_c(c_{1,1}, \ldots, c_{K,N})$: **return** 0
return 1

Fig. 4. The algorithm Check used in the issuing protocol of blind signature scheme $\mathsf{PIKA}_\mathsf{CDH}$, where $\mathsf{H} : \{0,1\}^* \to \mathbb{G}, \mathsf{H}' : \{0,1\}^* \to \{0,1\}^n$ and $\mathsf{H}_\mathsf{r}, \mathsf{H}_c : \{0,1\}^* \to \{0,1\}^n, \mathsf{H}_x : \{0,1\}^* \to \mathbb{Z}_p \times \mathcal{R}_\mathsf{ck} \times \{0,1\}^{n_\mathsf{PRF}}$ are random oracles.

4.3 Security Analysis

Completeness of the scheme follows by inspection. We show blindness and one-more unforgeability. For one-more unforgeability, we show $q_{\max}$-OMUF, where $q_{\max}$ is a parameter that can be set freely (e.g. $q_{\max} = 2^{30}$) and has influence the function f. We note that making f grow quadratically, one could show full OMUF using a similar proof.

Theorem 3. *Let* PRF *be a puncturable pseudorandom function and* CMT *be a randomness homomorphic commitment scheme. Let* $\mathsf{H}' : \{0,1\}^* \to \{0,1\}^n$ *and* $\mathsf{H}_\mathsf{r} : \{0,1\}^* \to \{0,1\}^n, \mathsf{H}_x : \{0,1\}^* \to \mathbb{Z}_p \times \mathcal{R}_\mathsf{ck} \times \{0,1\}^{n_\mathsf{PRF}}$ *be random oracles. Then* $\mathsf{PIKA}_\mathsf{CDH}$ *satisfies malicious signer blindness.*

In particular, for any adversary who uses N^L *and* N^R *as the counters in its executions with the user and queries* $\mathsf{H}', \mathsf{H}_\mathsf{r}, \mathsf{H}_x$ *at most* $Q_{\mathsf{H}'}, Q_{\mathsf{H}_\mathsf{r}}, Q_{\mathsf{H}_x}$ *times, respectively, the malicious signer blindness advantage can be bounded by*

$$4\epsilon_\mathsf{PRF} + \frac{Q_{\mathsf{H}'}^2}{2^{n-1}} + \frac{Q_{\mathsf{H}'}}{2^{n-2}} + \frac{KQ_{\mathsf{H}_x}}{2^{n_\mathsf{PRF}-2}} + \frac{KQ_{\mathsf{H}_\mathsf{r}}}{2^{n_\mathsf{PRF}-2}},$$

where ϵ_PRF *is the advantage of an adversary against the security of* PRF *with input length* $\max\{\log(N^L), \log(N^R)\}$ *when puncturing at* K *points.*

Due to space limitation, we postpone the proof to Supplementary Material Section I.

Theorem 4. *Let* CMT *be a randomness homomorphic commitment scheme and* PRF *be a puncturable pseudorandom function. Let* $\mathsf{PGGen}(1^n)$ *be a bilinear group*

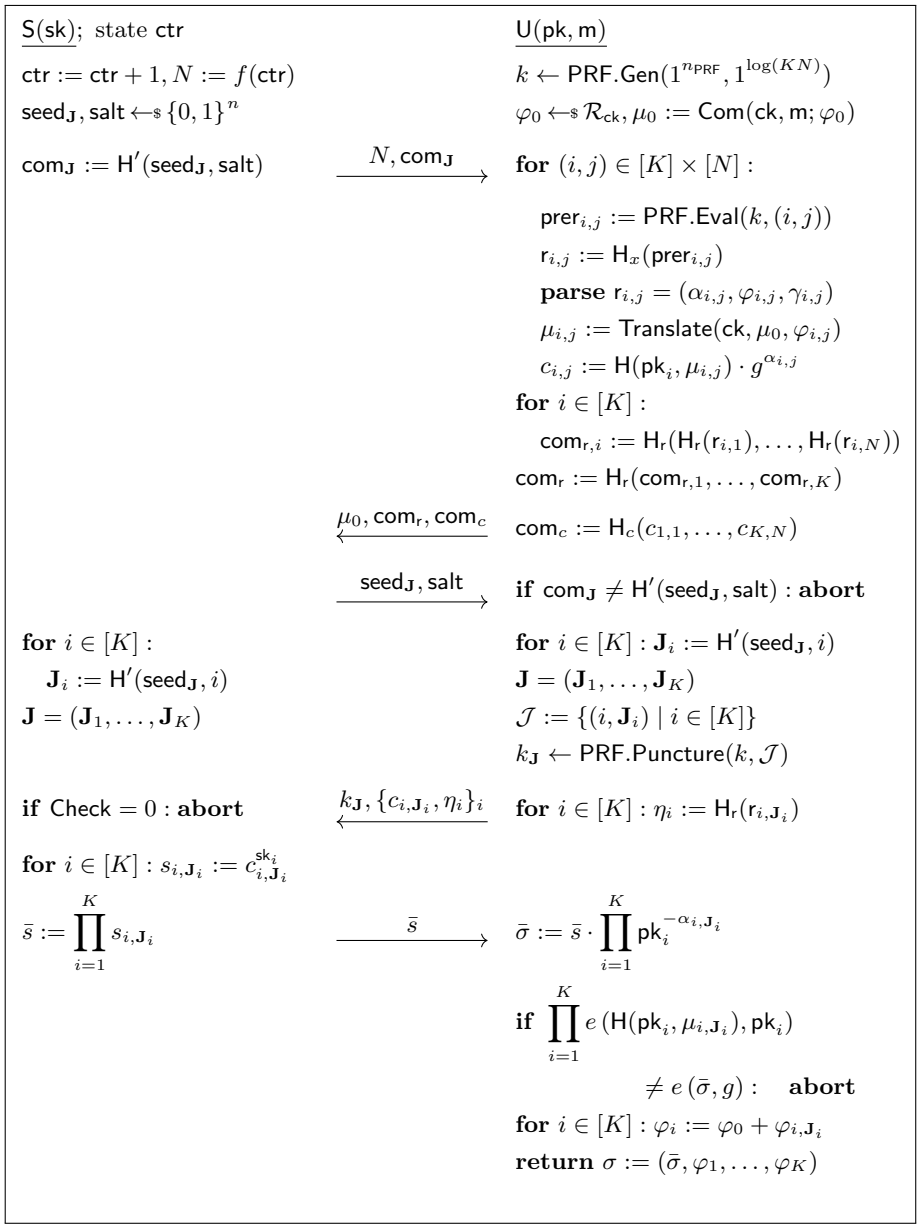

Fig. 5. The signature issuing protocol of the blind signature scheme $\mathsf{PIKA}_{\mathsf{CDH}}$, where $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_p, \mathsf{H}' : \{0,1\}^* \to \{0,1\}^n$ and $\mathsf{H}_\mathsf{r}, \mathsf{H}_c : \{0,1\}^* \to \{0,1\}^n, \mathsf{H}_x : \{0,1\}^* \to \mathbb{Z}_p \times \mathcal{R}_\mathsf{ck} \times \{0,1\}^{n_\mathsf{PRF}}$ are random oracles. The algorithm Check is defined in Figure 4. The state ctr of S is incremented atomically.

generation algorithm. Further, let $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_p, \mathsf{H}' : \{0,1\}^* \to \{0,1\}^n$ *and* $\mathsf{H}_\mathsf{r}, \mathsf{H}_c : \{0,1\}^* \to \{0,1\}^n$ *be random oracles. Also, assume that there is a* $\vartheta > 0$ *and* f *is such that*

$$f(\mathsf{ctr}) = \lceil 3\vartheta \ln(q_{\max} + 1) \cdot \mathsf{ctr} \rceil .$$

Then $\mathsf{PIKA}_{\mathsf{CDH}}$ *satisfies* $q_{\max}$*-one-more unforgeability, under the* CDH *assumption relative to* PGGen*.*

Specifically, assume the existence of an adversary against the OMUF *security of* $\mathsf{PIKA}_{\mathsf{CDH}}$ *that has advantage* ϵ*, runs in time* t*, makes at most* $Q_{\mathsf{H}_\mathsf{r}}, Q_{\mathsf{H}_c}, Q_{\mathsf{H}'}, Q_{\mathsf{H}}$ *queries to oracles* $\mathsf{H}_\mathsf{r}, \mathsf{H}_c, \mathsf{H}', \mathsf{H}$*, respectively, and starts at most* $q \leq q_{\max}$ *interactions with his signer oracle. Let* $\delta > 0$ *such that* $(1-\delta)\vartheta > 1$*. Then there exists an adversary against the* CDH *problem relative to* PGGen *with advantage* ϵ_{CDH} *and running time* t *and an adversary against the binding property of* CMT *with advantage* ϵ_{CMT} *and running time* t *such that*

$$\epsilon - e^{-\delta K} \leq \epsilon_{\mathsf{CMT}} + \frac{K}{p} + 4qK\epsilon_{\mathsf{CDH}} + \mathsf{stat}$$

where

$$\mathsf{stat} = \frac{Q_{\mathsf{H}_\mathsf{r}}^2}{2^n} + \frac{Q_{\mathsf{H}_c}^2}{2^n} + \frac{qQ_{\mathsf{H}_\mathsf{r}}}{2^n} + \frac{qKQ_{\mathsf{H}_\mathsf{r}}}{2^n} + \frac{qQ_{\mathsf{H}_c}}{2^n} + \frac{qQ_{\mathsf{H}'}}{2^{n-1}}.$$

Proof. Set $\mathsf{BS} := \mathsf{PIKA}_{\mathsf{CDH}}$. Let $\mathcal{A}$ be an adversary against the OMUF security of BS. We prove the statement via a sequence of games.

Game $\mathbf{G}_0$: We start with game $\mathbf{G}_0 := q_{\max}\text{-}\mathbf{OMUF}^{\mathcal{A}}_{\mathsf{BS}}$, which is the one-more unforgeability game. We briefly recall this game. A key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^n)$ is sampled, $\mathcal{A}$ is run with concurrent access to an interactive oracle O simulating the signer $\mathsf{S}(\mathsf{sk})$. Assume that $\mathcal{A}$ completes ℓ interactions with O. Further, $\mathcal{A}$ gets access to random oracles $\mathsf{H}, \mathsf{H}', \mathsf{H}_\mathsf{r}$ and H_c, which are provided by the game in the standard lazy manner. When $\mathcal{A}$ finishes its execution, it outputs tuples $(\mathsf{m}_1, \sigma_1), \ldots, (\mathsf{m}_k, \sigma_k)$ and wins, if all m_i are distinct, $k > \ell$ and all signatures σ_i verify with respect to pk and m_i.

Game $\mathbf{G}_1$: In game $\mathbf{G}_1$, we add an additional abort. The game aborts if in the end $\mathcal{A}$'s output contains two pairs $(\mathsf{m}^{(0)}, \sigma^{(0)}), (\mathsf{m}^{(1)}, \sigma^{(1)})$ such that $\mathsf{m}^{(0)} \neq \mathsf{m}^{(1)}$ but there exists $i^{(0)}, i^{(1)} \in [K]$ such that

$$\mathsf{Com}(\mathsf{ck}, \mathsf{m}^{(0)}; \varphi^{(0)}_{i^{(0)}}) = \mathsf{Com}(\mathsf{ck}, \mathsf{m}^{(1)}; \varphi^{(1)}_{i^{(1)}}).$$

As CMT is computationally binding, a straight-forward reduction with advantage ϵ_{CMT} and running time t shows that

$$|\Pr[\mathbf{G}_0 \Rightarrow 1] - \Pr[\mathbf{G}_1 \Rightarrow 1]| \leq \epsilon_{\mathsf{CMT}}.$$

Game $\mathbf{G}_2$: This game is as $\mathbf{G}_1$, but we rule out collisions for oracles $\mathsf{H}_t, t \in \{\mathsf{r}, c\}$.

To be more precise, we change the simulation of oracles $\mathsf{H}_t, t \in \{\mathsf{r}, c\}$ in the following way. If $\mathcal{A}$ queries $\mathsf{H}_t(x)$ and this value is not yet defined, the game samples an image $y \leftarrow\!\!\$\, \{0,1\}^n$. However, if there exists an $x' \neq x$ with $\mathsf{H}_t(x') = y$, the game returns $\perp$. Otherwise it behaves as before. Note that $\mathcal{A}$ can only distinguish between $\mathbf{G}_1$ and $\mathbf{G}_2$ if such a collision happens, i.e. H_t returns $\perp$. We can apply a union bound over all $Q^2_{\mathsf{H}_t}$ pairs of random oracle queries and obtain

$$|\Pr[\mathbf{G}_1 \Rightarrow 1] - \Pr[\mathbf{G}_2 \Rightarrow 1]| \leq \frac{Q^2_{\mathsf{H}_\mathsf{r}}}{2^n} + \frac{Q^2_{\mathsf{H}_c}}{2^n}.$$

Note that the change in $\mathbf{G}_2$ implies that at each point of the execution of the game and for each image $y \in \{0,1\}^n$, there is at most one preimage $\mathsf{H}_t^{-1}(y)$ under H_t. By looking at the random oracle queries of $\mathcal{A}$, the game can extract preimages of given images y, and we know that for each y at most one preimage can be extracted. We will make use of such an extraction in the following games.

Game $\mathbf{G}_3$: We change the way the signer oracle is executed. In particular, when $\mathcal{A}$ sends $\mu_0, \mathsf{com}_\mathsf{r}, \mathsf{com}_c$ as its first message, the game tries to extract values $\bar{\mathsf{com}}_{\mathsf{r},i}$ such that $\mathsf{com}_\mathsf{r} = \mathsf{H}_\mathsf{r}(\bar{\mathsf{com}}_{\mathsf{r},1}, \ldots, \bar{\mathsf{com}}_{\mathsf{r},K})$ by searching through random oracle queries. If the game can not extract such a preimage, we write $\bar{\mathsf{com}}_{\mathsf{r},i} = \perp$ for all $i \in [K]$. Then, the game aborts if it can not extract such a preimage , i.e. $\bar{\mathsf{com}}_{\mathsf{r},i} = \perp$, but later algorithm Check outputs 1. Recall that algorithm Check verifies that

$$\mathsf{com}_\mathsf{r} = \mathsf{H}_\mathsf{r}(\mathsf{com}_{\mathsf{r},1}, \ldots, \mathsf{com}_{\mathsf{r},K}).$$

Thus, for every fixed interaction, we can bound the probability of such an abort by $Q_{\mathsf{H}_\mathsf{r}}/2^n$. Indeed, once com_r is sent by $\mathcal{A}$ and thus fixed, and the game can not extract, we know that there is no bitstring x such that $\mathsf{H}_\mathsf{r}(x) = \mathsf{com}_\mathsf{r}$. Also, if algorithm Check outputs 1, we know that $\mathcal{A}$ was able to find a preimage of com_r after this was fixed. This can happen with probability at most $1/2^n$ for each random oracle query. Using a union bound over all interactions we obtain

$$|\Pr[\mathbf{G}_3 \Rightarrow 1] - \Pr[\mathbf{G}_4 \Rightarrow 1]| \leq \frac{qQ_{\mathsf{H}_\mathsf{r}}}{2^n}.$$

Game $\mathbf{G}_4$: We introduce another abort in the signer oracle. In this game, after the extraction of $(\bar{\mathsf{com}}_{\mathsf{r},1}, \ldots, \bar{\mathsf{com}}_{\mathsf{r},K})$ from com_r we introduced before, the game extracts $(\bar{\mathsf{r}}_{i,1}, \ldots, \bar{\mathsf{r}}_{i,N})$ from $\bar{\mathsf{com}}_{\mathsf{r},i}$ for every $i \in [K]$ for which $\bar{\mathsf{com}}_{\mathsf{r},i} \neq \perp$, such that

$$\bar{\mathsf{com}}_{\mathsf{r},i} = \mathsf{H}_\mathsf{r}(\mathsf{H}_\mathsf{r}(\bar{\mathsf{r}}_{i,1}), \ldots, \mathsf{H}_\mathsf{r}(\bar{\mathsf{r}}_{i,N})).$$

Again, the game does this by looking at the random oracle queries of $\mathcal{A}$ and we write $\bar{\mathsf{r}}_{i,j} = \perp$ if the game can not extract the value $\bar{\mathsf{r}}_{i,j}$. If there is an instance $i \in [K]$ and a session $j \in [N]$ such that $\bar{\mathsf{com}}_{\mathsf{r},i} \neq \perp$ but $\bar{\mathsf{r}}_{i,j} = \perp$ and later in that execution $\mathbf{J}_i \neq j$ but algorithm Check outputs 1, the game aborts.

To analyze the probability of this abort, fix an interaction and an instance $i \in [K]$. Assume that $\bar{\mathsf{com}}_{\mathsf{r},i} \neq \perp$ and there is a session $j \in [N]$ such that $\bar{\mathsf{r}}_{i,j} = \perp$ and later in that interaction $\mathbf{J}_i \neq j$. Then, after $\bar{\mathsf{com}}_{\mathsf{r},i}$ is fixed, we

consider two cases. In the first case, the game could not extract $h_1, \ldots, h_N$ such that $\bar{\mathsf{com}}_{\mathsf{r},i} = \mathsf{H}_{\mathsf{r}}(h_1, \ldots, h_N)$. Clearly, once $\bar{\mathsf{com}}_{\mathsf{r},i}$, the probability that one of the hash queries of $\mathcal{A}$ evaluates to $\bar{\mathsf{com}}_{\mathsf{r},i}$ is at most $1/2^n$. Thus, the probability that Check outputs 1, i.e. $\mathcal{A}$ is able to open $\bar{\mathsf{com}}_{\mathsf{r},i}$ in this case, is at most $Q_{\mathsf{H}_{\mathsf{r}}}/2^n$. Similarly, in the case where the game could extract $h_1, \ldots, h_N$, but could not extract $\bar{\mathsf{r}}_{i,j}$ such that $\mathsf{H}_{\mathsf{r}}(\bar{\mathsf{r}}_{i,j}) = h_j$, the probability that one of $\mathcal{A}$'s hash queries evaluates to h_j is at most $1/2^n$. Thus, the probability that Check outputs 1, i.e. $\mathcal{A}$ is able to open h_j in this case, is at most $Q_{\mathsf{H}_{\mathsf{r}}}/2^n$. Note that here we needed that $j \neq \mathbf{J}_i$, as the definition of Check does not require $\mathcal{A}$ to open $h_{\mathbf{J}_i}$.

Applying a union bound over the interactions and instances we get

$$|\Pr[\mathbf{G}_3 \Rightarrow 1] - \Pr[\mathbf{G}_4 \Rightarrow 1]| \leq \frac{qKQ_{\mathsf{H}_{\mathsf{r}}}}{2^n}.$$

Game $\mathbf{G}_5$: We introduce another abort: Whenever $\mathcal{A}$ sends $\mu_0, \mathsf{com}_{\mathsf{r}}, \mathsf{com}_c$ as its first message, the game behaves as before, but additionally the game extracts values $\bar{c}_{1,1}, \ldots, \bar{c}_{K,N}$ from com_c such that

$$\mathsf{com}_c = \mathsf{H}_c(\bar{c}_{1,1}, \ldots, \bar{c}_{K,N}).$$

If the game can not extract, but later algorithm Check outputs 1, the game aborts. Note that algorithm Check internally checks if

$$\mathsf{com}_c = \mathsf{H}_c(c_{1,1}, \ldots, c_{K,N}).$$

Thus, for each fixed interaction it is possible to argue as in the previous games to bound the probability of such an abort and hence we obtain

$$|\Pr[\mathbf{G}_4 \Rightarrow 1] - \Pr[\mathbf{G}_5 \Rightarrow 1]| \leq \frac{qQ_{\mathsf{H}_c}}{2^n}.$$

Game $\mathbf{G}_6$: In $\mathbf{G}_6$, the signer oracle sends a random $\mathsf{com}_{\mathbf{J}}$ in the beginning of each interaction. Later, before it has to send $\mathsf{seed}_{\mathbf{J}}, \mathsf{salt}$, it samples $\mathsf{salt} \leftarrow\!\!\$\, \{0,1\}^n$ and aborts if $\mathsf{H}'(\mathsf{seed}_{\mathbf{J}}, \mathsf{salt})$ is already defined. If it is not yet defined, it defines it as $\mathsf{H}'(\mathsf{seed}_{\mathbf{J}}, \mathsf{salt}) := \mathsf{com}_{\mathbf{J}}$. The adversary $\mathcal{A}$ can only distinguish between $\mathbf{G}_5$ and $\mathbf{G}_6$ if $\mathsf{H}'(\mathsf{seed}_{\mathbf{J}}, \mathsf{salt})$ is already defined. By a union bound over all $Q_{\mathsf{H}'}$ hash queries and q interactions we obtain

$$|\Pr[\mathbf{G}_5 \Rightarrow 1] - \Pr[\mathbf{G}_6 \Rightarrow 1]| \leq \frac{qQ_{\mathsf{H}'}}{2^n}.$$

Game $\mathbf{G}_7$: In $\mathbf{G}_7$, the game aborts if in some interaction there exists an $i \in [K]$ such that $\mathsf{H}'(\mathsf{seed}_{\mathbf{J}}, i)$ has already been queried before the signing oracle sends $\mathsf{seed}_{\mathbf{J}}$ to $\mathcal{A}$. Clearly, $\mathcal{A}$ obtains no information about $\mathsf{seed}_{\mathbf{J}}$ before the potential abort, see $\mathbf{G}_6$. Further, $\mathsf{seed}_{\mathbf{J}}$ is sampled uniformly at random. A union bound over all $Q_{\mathsf{H}'}$ queries and q interactions shows that

$$|\Pr[\mathbf{G}_6 \Rightarrow 1] - \Pr[\mathbf{G}_7 \Rightarrow 1]| \leq \frac{qQ_{\mathsf{H}'}}{2^n}.$$

Now, fix an interaction in $\mathbf{G}_7$ and assume that Check returns 1 and the game does not abort due to any of the reasons we introduced so far. Note that this means that for all instances $i \in [K]$ the value $\bar{\mathsf{com}}_{\mathsf{r},i}$ could be extracted. Furthermore, this means that if there exists $i \in [K], j_0 \in [N]$ such that $\bar{\mathsf{r}}_{i,j_0} = \perp$ then later $\mathbf{J}_i = j_0$. Also, note that if Check does not abort, then we have $\bar{\mathsf{com}}_{\mathsf{r},i} = \mathsf{com}_{\mathsf{r},i}, \bar{\mathsf{r}}_{i,j} = \mathsf{r}_{i,j}$ and $\bar{c}_{i,j} = c_{i,j}$ for all $(i,j) \in [K] \times [N]$ for which these values are defined. This is because we ruled out collisions for oracles $\mathsf{H}_\mathsf{r}, \mathsf{H}_c$. Now, we define an indicator random variable $\mathsf{cheat}_{i,\mathsf{ctr}}$ for the event that in the ctr^{th} interaction, the signer oracle does not abort and there exists $i \in [K], j \in [N]$ such that $\bar{\mathsf{r}}_{i,j} = \perp$ or $\bar{\mathsf{r}}_{i,j} = (\alpha, \varphi, \gamma)$ such that

$$c_{i,j} \neq \mathsf{H}(\mathsf{pk}_i, \mathsf{Translate}(\mathsf{ck}, \mu_0, \varphi)) \cdot g^{\alpha}.$$

We say that $\mathcal{A}$ successfully cheats in instance $i \in [K]$ and interaction ctr if $\mathsf{cheat}_{i,\mathsf{ctr}} = 1$. We also define the number of interactions in which $\mathcal{A}$ successfully cheats in instance i as $\mathsf{cheat}_i^* := \sum_{\mathsf{ctr}=2}^{q+1} \mathsf{cheat}_{i,\mathsf{ctr}}$.

By the above discussion, we have that $\mathsf{cheat}_{i,\mathsf{ctr}} = 1$ implies that $\mathbf{J}_i = j_0$ and thus

$$\Pr\left[\mathsf{cheat}_{i,\mathsf{ctr}} = 1\right] \leq \frac{1}{N}.$$

Therefore, we can bound the expectation of cheat_i^* using

$$\mathbb{E}\left[\mathsf{cheat}_i^*\right] \leq \frac{1}{3\vartheta \ln(q_{\max}+1)} \sum_{\mathsf{ctr}=2}^{q+1} \frac{1}{\mathsf{ctr}} \leq \frac{\ln(q+1)}{3\vartheta \ln(q_{\max}+1)} \leq \frac{1}{3\vartheta}.$$

Now, if we plug $X := \mathsf{cheat}_i^*$ and $s := 3\mathbb{E}\left[\mathsf{cheat}_i^*\right] + \delta = 1/\vartheta + \delta$ into the Chernoff bound (Lemma 5), we get that for all $i \in [K]$

$$\Pr\left[\mathsf{cheat}_i^* \geq \frac{1}{\vartheta} + \delta\right] \leq e^{-\delta}.$$

We note that the entire calculation of this probability also holds if we fix the random coins of the adversary.

Game $\mathbf{G}_8$: Game $\mathbf{G}_8$ is defined as $\mathbf{G}_7$, but additionally aborts if for all $i \in [K]$ we have $\mathsf{cheat}_i^* \geq \delta + 1/\vartheta$. In particular, if $\mathbf{G}_8$ does not abort, then there is some instance i for which $\mathcal{A}$ does not successfully cheat at all, which follows from the assumption $(1-\delta)\vartheta > 1$.

We can now bound the distinguishing advantage of $\mathcal{A}$ between $\mathbf{G}_7$ and $\mathbf{G}_8$ as follows. We denote the random coins of $\mathcal{A}$ by $\rho_\mathcal{A}$ and the random coins of the experiment (excluding $\rho_\mathcal{A}$) by ρ. Let bad be the event that for all $i \in [K]$ we have $\mathsf{cheat}_i^* \geq \delta + 1/\vartheta$. We note that the coins ρ that the experiment uses for

the K instances are independent. Thus we have

$$\begin{aligned}\Pr_{\rho,\rho_{\mathcal{A}}}[\mathsf{bad}] &= \sum_{\bar{\rho}_{\mathcal{A}}} \Pr_{\rho_{\mathcal{A}}}[\rho_{\mathcal{A}} = \bar{\rho}_{\mathcal{A}}] \cdot \Pr_{\rho,\rho_{\mathcal{A}}}[\mathsf{bad} \mid \rho_{\mathcal{A}} = \bar{\rho}_{\mathcal{A}}] \\ &= \sum_{\bar{\rho}_{\mathcal{A}}} \Pr_{\rho_{\mathcal{A}}}[\rho_{\mathcal{A}} = \bar{\rho}_{\mathcal{A}}] \cdot \prod_{i\in[K]} \Pr_{\rho,\rho_{\mathcal{A}}}\left[\mathsf{cheat}_i^* \geq \frac{1}{\vartheta} + \delta \;\middle|\; \rho_{\mathcal{A}} = \bar{\rho}_{\mathcal{A}}\right] \\ &\leq \sum_{\bar{\rho}_{\mathcal{A}}} \Pr_{\rho_{\mathcal{A}}}[\rho_{\mathcal{A}} = \bar{\rho}_{\mathcal{A}}] \cdot e^{-\delta K} = e^{-\delta K},\end{aligned}$$

which implies

$$|\Pr[\mathbf{G}_7 \Rightarrow 1] - \Pr[\mathbf{G}_8 \Rightarrow 1]| \leq \Pr_{\rho,\rho_{\mathcal{A}}}[\mathsf{bad}] \leq e^{-\delta K}.$$

Game $\mathbf{G}_9$: In game $\mathbf{G}_9$, we sample a random instance $i^* \leftarrow\!\!\$\, [K]$ at the beginning of the game. In the end, the game aborts if $\mathsf{cheat}_{i^*}^* \geq \delta + 1/\vartheta$. In particular, if this game does not abort, then $\mathcal{A}$ does not successfully cheat in instance i^* at all. As $\mathcal{A}$'s view is independent from i^*, we have

$$\begin{aligned}\Pr[\mathbf{G}_9 \Rightarrow 1] &= \Pr\left[\mathbf{G}_8 \Rightarrow 1 \wedge \mathsf{cheat}_{i^*}^* < \frac{1}{\vartheta} + \delta\right] \\ &= \Pr[\mathbf{G}_8 \Rightarrow 1] \cdot \Pr\left[\mathsf{cheat}_{i^*}^* < \frac{1}{\vartheta} + \delta \;\middle|\; \mathbf{G}_8 \Rightarrow 1\right] \\ &\geq \Pr[\mathbf{G}_8 \Rightarrow 1] \cdot \Pr\left[\mathsf{cheat}_{i^*}^* < \frac{1}{\vartheta} + \delta \;\middle|\; \exists i \in [K] : \mathsf{cheat}_i^* < \frac{1}{\vartheta} + \delta\right] \\ &\geq \Pr[\mathbf{G}_8 \Rightarrow 1] \cdot \frac{1}{K},\end{aligned}$$

where the first inequality follows from the fact that the event $\mathbf{G}_8 \Rightarrow 1$ implies the event $\exists i \in [K] : \mathsf{cheat}_i^* < \delta + 1/\vartheta$.

We note that from now on, our proof follows the proof strategy of the BLS signature scheme [5].

Game $\mathbf{G}_{10}$: In game $\mathbf{G}_{10}$, we introduce an initially empty set $\mathcal{L}$ and a new abort. We highlight that we treat $\mathcal{L}$ as a set and therefore every bitstring is in $\mathcal{L}$ only once. Recall that when $\mathcal{A}$ sends $\mu_0, \mathsf{com}_{\mathsf{r}}, \mathsf{com}_c$ to the signer oracle, the game tries to extract values $\bar{\mathsf{r}}_{i,j}$ for $(i,j) \in [K] \times [N]$. Then the game samples $\mathsf{seed}_{\mathbf{J}}$ and computes $\mathbf{J}$ accordingly. In particular, due to the changes in the previous games we know that the game extracts $\bar{\mathsf{r}}_{i^*,\mathbf{J}_{i^*}} = (\alpha, \varphi, \gamma)$ unless the experiment will abort anyways. Then, in game $\mathbf{G}_{10}$, the game will insert $\mathsf{Translate}(\mathsf{ck}, \mu_0, \varphi)$ into $\mathcal{L}$.

Fix the first pair (m, σ) in $\mathcal{A}$'s final output such that for $\sigma = (\bar{\sigma}, \varphi_1, \ldots, \varphi_K)$ and $\mu^* := \mathsf{Com}(\mathsf{ck}, \mathsf{m}; \varphi_{i^*})$ we have $\mu^* \notin \mathcal{L}$. Such a pair must exists if $\mathcal{A}$ is successful, see game $\mathbf{G}_1$. Then game $\mathbf{G}_{10}$ aborts if $\mathsf{H}(\mathsf{pk}_{i^*}, \mu^*)$ is not defined yet. Note that $\mathcal{A}$'s success probability in such a case can be at most $1/p$ and hence

$$|\Pr[\mathbf{G}_9 \Rightarrow 1] - \Pr[\mathbf{G}_{10} \Rightarrow 1]| \leq \frac{1}{p}.$$

Game $\mathbf{G}_{11}$: In game $\mathbf{G}_{11}$, we change how the random oracle H is simulated and add a new abort. For every query of the form $\mathsf{H}(\mathsf{pk}_{i^*}, \mu)$ the game independently samples a bit $b[\mu] \in \{0,1\}$ such that the probability that $b[\mu] = 1$ is $1/(q+1)$. Whenever the game adds a value μ to the set $\mathcal{L}$, it aborts if $b[\mu] = 1$. Then, after $\mathcal{A}$ returns its final output, the game determines μ^* as in $\mathbf{G}_{10}$, adds arbitrary values to $\mathcal{L}$ such that all values in $\mathcal{L} \cup \{\mu^*\}$ are distinct and $|\mathcal{L}| = q$ and aborts if $b[\mu^*] = 0$ or there is a $\mu \in \mathcal{L}$ such that $b[\mu] = 1$. Otherwise it continues as before. Note that unless the game aborts, $\mathcal{A}$'s view does not change. As all bits $b[\mu]$ are independent, we derive

$$\begin{aligned}
\Pr[\mathbf{G}_{11} \Rightarrow 1] &= \Pr[\mathbf{G}_{10} \Rightarrow 1] \cdot \Pr[b[\mu^*] = 1 \wedge \forall \mu \in \mathcal{L} : b[\mu] = 0] \\
&= \Pr[\mathbf{G}_{10} \Rightarrow 1] \cdot \frac{1}{q+1}\left(1 - \frac{1}{q+1}\right)^q \\
&= \Pr[\mathbf{G}_{10} \Rightarrow 1] \cdot \frac{1}{q}\left(1 - \frac{1}{q+1}\right)^{q+1} \\
&\geq \Pr[\mathbf{G}_{10} \Rightarrow 1] \cdot \frac{1}{4q},
\end{aligned}$$

where the last inequality follows from $(1 - 1/x)^x \geq 1/4$ for all $x \geq 2$.

Finally, we construct a reduction $\mathcal{B}$ that solves CDH with running time t and advantage ϵ_{CDH} such that

$$\Pr[\mathbf{G}_{11} \Rightarrow 1] \leq \epsilon_{\mathsf{CDH}}.$$

Then, the statement follows by an easy calculation. Reduction $\mathcal{B}$ works as follows:

- $\mathcal{B}$ gets as input bilinear group parameters $\mathbb{G}, g, p, e$ and group elements $X = g^x, Y = g^y$. The goal of $\mathcal{B}$ is to compute g^{xy}. First, $\mathcal{B}$ samples $i^* \leftarrow\!\!\$\, [K]$. Then, it defines $\mathsf{pk}_{i^*} := X$ (which implicitly defines $\mathsf{sk}_{i^*} := x$) and $\mathsf{sk}_i \leftarrow\!\!\$\, \mathbb{Z}_p, \mathsf{pk}_i := g^{\mathsf{sk}_i}$ for $i \in [K] \setminus \{i^*\}$.
- $\mathcal{B}$ runs adversary $\mathcal{A}$ on input $\mathbb{G}, g, p, e, \mathsf{pk} := (\mathsf{pk}_1, \ldots, \mathsf{pk}_K, \mathsf{ck})$ with oracle access to a signer oracle and random oracles $\mathsf{H}, \mathsf{H}_\mathsf{r}, \mathsf{H}_c, \mathsf{H}'$. To do so, it simulates oracles $\mathsf{H}_\mathsf{r}, \mathsf{H}_c, \mathsf{H}'$ exactly as in $\mathbf{G}_{11}$. The other oracles are provided as follows:
  - For a query of the form $\mathsf{H}(\mathsf{pk}_{i^*}, \mu)$ for which the hash value is not yet defined, it samples a bit $b[\mu] \in \{0,1\}$ such that the probability that $b[\mu] = 1$ is $1/(q+1)$. Then, it defines the hash value as $Y^{b[\mu]} \cdot g^{t[i^*,\mu]}$ for a randomly sampled $t[i^*, \mu] \leftarrow\!\!\$\, \mathbb{Z}_p$. For a query of the form $\mathsf{H}(\mathsf{pk}_i, \mu), i \neq i^*$ for which the hash value is not yet defined it defines the hash value as $g^{t[i,\mu]}$ for a randomly sampled $t[i, \mu] \leftarrow\!\!\$\, \mathbb{Z}_p$. For all other queries it simulates H honestly.
  - When $\mathcal{A}$ starts an interaction with the signer oracle, $\mathcal{B}$ sends N to $\mathcal{B}$ as in the protocol. When $\mathcal{B}$ sends its first message $\mu_0, \mathsf{com}_\mathsf{r}, \mathsf{com}_c$ as its first message, $\mathcal{B}$ behaves as $\mathbf{G}_{11}$. In particular, it tries to extract $\bar{\mathsf{r}}_{i,j}, \bar{c}_{i,j}$ for $(i,j) \in [K] \times [N]$. It then sends $\mathsf{seed}_\mathbf{J}$ to $\mathcal{A}$.

- When $\mathcal{A}$ sends its second message $k_{\mathbf{J}}, \{c_{i,\mathbf{J}_i}, \eta_i\}_{i\in[K]}$, $\mathcal{B}$ aborts under the same conditions as $\mathbf{G}_{11}$ does. In particular, if $\mathcal{B}$ does not abort and the signer oracle does not abort then $\bar{\mathsf{r}}_{i^*,\mathbf{J}_{i^*}} = (\alpha, \varphi, \gamma)$ is defined and $\mathcal{B}$ for $\mu := \mathsf{Translate}(\mathsf{ck}, \mu_0, \varphi)$, $\mathcal{B}$ sets $s_{i^*,\mathbf{J}_{i^*}} := X^{t[i^*,\mu]+\alpha}$. As defined in $\mathbf{G}_{11}$, $\mathcal{B}$ also inserts μ into the set $\mathcal{L}$. It computes $s_{i,\mathbf{J}_i}$ for $i \neq i^*$ as game $\mathbf{G}_{11}$ does, which is possible as $\mathcal{B}$ holds the corresponding sk_i. Then, $\mathcal{B}$ sends $\bar{s} := \prod_{i=1}^{K} s_{i,\mathbf{J}_i}$ to $\mathcal{A}$.

– When $\mathcal{A}$ returns its final output, $\mathcal{B}$ performs all verification steps in $\mathbf{G}_{11}$. In particular, it searches for the first pair (m, σ) in $\mathcal{A}$'s final output such that for $\sigma = (\bar{\sigma}, \varphi_1, \ldots, \varphi_K)$ and $\mu^* := \mathsf{Com}(\mathsf{ck}, \mathsf{m}; \varphi_{i^*})$ we have $\mu^* \notin \mathcal{L}$. As defined in $\mathbf{G}_{11}$, $\mathcal{B}$ aborts if $b[\mu^*] = 0$. Finally, $\mathcal{B}$ defines $\mu_i := \mathsf{Com}(\mathsf{ck}, \mathsf{m}; \varphi_i)$ and returns

$$Z := \bar{\sigma} \cdot X^{-t[i^*,\mu^*]} \cdot g^{-\sum_{i\in[K]\setminus\{i^*\}} t[i,\mu_i]\mathsf{sk}_i}$$

to its challenger.

We first argue that $\mathcal{B}$ perfectly simulates $\mathbf{G}_{11}$ for $\mathcal{A}$. To see that, note that as the $t[i,\mu]$ are sampled uniformly at random, the random oracle is simulated perfectly. To see that $s_{i^*,\mathbf{J}_{i^*}}$ is distributed correctly, note that if the signing oracle and $\mathbf{G}_{11}$ do not abort, then we have

$$c_{i^*,\mathbf{J}_{i^*}}^{\mathsf{sk}_{i^*}} = (\mathsf{H}(\mathsf{pk}_{i^*}, \mu) \cdot g^{\alpha})^{\mathsf{sk}_{i^*}} = \left(Y^{b[\mu]} \cdot g^{t[i^*,\mu]} \cdot g^{\alpha}\right)^{x} = X^{t[i^*,\mu]+\alpha},$$

where the last equality follows from $b[\mu] = 0$, as otherwise $\mathbf{G}_{11}$ would have aborted.

It remains to show that if $\mathbf{G}_{11}$ outputs 1, then we have $Z = g^{xy}$. This follows directly from the verification equation and $b[\mu^*] = 1$. To see this, note that

$$\begin{aligned}\prod_{i=1}^{K} e\left(\mathsf{H}(\mathsf{pk}_i, \mu_i), \mathsf{pk}_i\right) &= e\left(Y^{b[\mu^*]} \cdot g^{t[i^*,\mu^*]}, X\right) \cdot \prod_{i\in[K]\setminus\{i^*\}} e\left(g^{t[i,\mu_i]}, g^{\mathsf{sk}_i}\right) \\ &= e\left(g,g\right)^{xy+t[i^*,\mu^*]x} \cdot e\left(g,g\right)^{\sum_{i\in[K]\setminus\{i^*\}} t[i,\mu_i]\mathsf{sk}_i}.\end{aligned}$$

Using the verification equation, this implies that

$$g^{xy} = \bar{\sigma} \cdot g^{-\left(t[i^*,\mu^*]x + \sum_{i\in[K]\setminus\{i^*\}} t[i,\mu_i]\mathsf{sk}_i\right)}$$

Concluded. □

We note that instead of giving games $\mathbf{G}_{10}, \mathbf{G}_{11}$ and the reduction from CDH explicitly, one can also directly reduce from the security of the BLS signature scheme to $\mathbf{G}_9$, leading to the very same bound in total. This tells us that one can use (up to losing $\log(K)$ bits [6] of security) the same curves as for BLS.

Corollary 1 (Informal). *Under the same conditions as in Theorem 4, the scheme* $\mathsf{PIKA}_{\mathsf{CDH}}$ *satisfies* $q_{\max}$*-one-more unforgeability, if the BLS signature scheme [5] is unforgeable under chosen message attacks relative to* PGGen*, where the concrete security loss is (up to statistically negligible terms) given by* K*.*

[6] In our concrete instantiation, $\log(K) \approx 6.5$.

4.4 Concrete Parameters and Efficiency

Let us now discuss concrete parameters for our scheme $\mathsf{PIKA}_{\mathsf{CDH}}$ based on the CDH assumption. Recall that the scheme uses parameters K, ϑ and p. Instantiating the commitment scheme CMT with a Pedersen commitment we also have to set a value for the order p' of the group that is used in this commitment scheme. Say that we aim for κ bits of security. In particular, we want to find appropriate values for $K, \vartheta, |p|$ and $|p'|$. Consider an adversary with running time t and advantage ϵ against the OMUF security of the scheme. If $\epsilon/t < 2^{-\kappa}$ we are done. Otherwise we have $\epsilon/t \geq 2^{-\kappa}$ and $\epsilon \geq 2^{-\kappa}$, as $t \geq 1$. Now, we want to use Theorem 4 to end up with a contradiction. If we use Theorem 4 with $\delta := -\ln(\epsilon/2)/K$, then $e^{-\delta K} = \epsilon/2$ and the security bound becomes

$$\epsilon \leq 2\left(\epsilon_{\mathsf{CMT}} + \frac{K}{p} + 4qK\epsilon_{\mathsf{CDH}} + \mathsf{stat}\right).$$

Assuming κ_{CDH} bits of security for the CDH instance and κ_{CMT} bits of security for the commitment scheme CMT we obtain

$$\epsilon \leq 2\left(2^{-\kappa_{\mathsf{CMT}}} \cdot t + \frac{K}{p} + 4qK \cdot 2^{-\kappa_{\mathsf{CDH}}} \cdot t + \mathsf{stat}\right).$$

We can now increase κ_{CDH} and κ_{CMT} (for a fixed combination of ϵ and t) until this inequality does not hold anymore. Then the adversary could not have existed in the first place. Using κ_{CDH} and κ_{CMT}, we can then determine an appropriate choice for $|p| = 2\kappa_{\mathsf{CDH}} + 1$ and $|p'| = 2\kappa_{\mathsf{CMT}} + 1$, see [37].

However, note that we can only apply this approach, if $(1-\delta)\vartheta > 1$, due to Theorem 4. By our choice of δ it is therefore sufficient to guarantee that $\left(1 - \ln(2^{\kappa+1})/K\right)\vartheta > 1$. It is clear that for a decreasing K, we have to increase ϑ to satisfy this constraint. Thus, our approach is as follows: For a few choices of K, we determine the minimum $\vartheta > 0$, such that the constraint holds. If there is no such ϑ, we throw away this particular K. Then, we proceed as discussed above to find security levels for the underlying instances and compute the signature sizes and key sizes.

Next, we focus on blindness. For simplicity, assume that $N^L = N^R =: N$. We instantiate PRF using a GGM construction with a random oracle $\mathsf{H}_{\mathsf{PRF}}$ (cf. Supplementary Material Section E) and know that $\epsilon_{\mathsf{PRF}} \leq (2\log(NK) - 1)KQ_{\mathsf{H}_{\mathsf{PRF}}}/2^{n_{\mathsf{PRF}}}$, where n_{PRF} is the output length of the pseudorandom function. By applying Theorem 3 we obtain the security bound

$$\frac{(2\log(N) + 2\log(K) - 1)\,KQ_{\mathsf{H}_{\mathsf{PRF}}}}{2^{n_{\mathsf{PRF}}-2}} + \frac{Q_{\mathsf{H}'}^2}{2^{n-1}} + \frac{Q_{\mathsf{H}'}}{2^{n-2}} + \frac{Q_{\mathsf{H}_x}}{2^{n_{\mathsf{PRF}}-2}} + \frac{KQ_{\mathsf{H}_{\mathsf{r}}}}{2^{n_{\mathsf{PRF}}-2}},$$

where n_{PRF} denotes the output length of PRF. Thus, we only have to increase n_{PRF} until the security bound guarantees κ bit of security.

We implemented the approach discussed above in Python script, see Supplementary Material Section K.2. To simplify a bit, we made the conservative assumption that the number of hash queries for each random oracle is equal to

the running time of the adversary and set N^L and N^R in the blindness bound to be equal to the maximum number q of signatures interactions that the adversary starts. Results can be found in Table 1.

5 A Concrete Scheme based on RSA

In addition to our concrete scheme from CDH, we also construct a concrete scheme $\mathsf{BS}_{\mathsf{RSA}}$ based on the RSA assumption. We postpone the details to Supplementary Material Section H and only give a short overview here.

Our scheme is based on the Okamoto-Guillou-Quisquater (OGQ) [31] linear function. That is, we start with this function in our generic transformation from Section 3. Informally, the function has domain $\mathcal{D} := \mathbb{Z}_\lambda \times \mathbb{Z}_N^*$, scalar space $\mathcal{S} := \mathbb{Z}_\lambda$ and range $\mathbb{Z}_N^*$, where N is an RSA modulus and λ is a prime with $\gcd(N, \lambda) = \gcd(\varphi(N), \lambda) = 1$. As we can not aggregate signatures efficiently, we can not mimic the K-repetition technique from our CDH-based scheme. Thus, we still rely on the loose bound of the underlying linear blind signature scheme. To solve this issue and obtain practical parameter sizes, we note that the bound becomes acceptable, once we increase the parameter λ. Our insight is that this can be done independently from the modulus N.

Although this improves the bound and thus concrete parameters, we still have a rather large communication complexity, due to the logarithmic number of $R_i \in \mathbb{Z}_N^*$ that are sent in our generic transformation. Here, our solution is to send a short random seed (e.g. 128 bit) and derive the values R_i using a random oracle. Now, the signer has to recover the preimages of the R_i to continue the protocol. We show that the OGQ linear function admits a trapdoor, that allows to sample preimages, solving this problem as well.

References

1. Agrawal, S., Kirshanova, E., Stehlé, D., Yadav, A.: Can round-optimal lattice-based blind signatures be practical? Cryptology ePrint Archive, Report 2021/1565 (2021), `https://eprint.iacr.org/2021/1565`
2. Bellare, M., Namprempre, C., Pointcheval, D., Semanko, M.: The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. Journal of Cryptology 16(3), 185–215 (Jun 2003)
3. Benhamouda, F., Lepoint, T., Loss, J., Orrù, M., Raykova, M.: On the (in)security of ROS. In: Canteaut, A., Standaert, F.X. (eds.) Advances in Cryptology – EUROCRYPT 2021, Part I. Lecture Notes in Computer Science, vol. 12696, pp. 33–53. Springer, Heidelberg, Germany, Zagreb, Croatia (Oct 17–21, 2021)
4. Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In: Desmedt, Y. (ed.) PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography. Lecture Notes in Computer Science, vol. 2567, pp. 31–46. Springer, Heidelberg, Germany, Miami, FL, USA (Jan 6–8, 2003)
5. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. In: Boyd, C. (ed.) Advances in Cryptology – ASIACRYPT 2001. Lecture Notes in

Computer Science, vol. 2248, pp. 514–532. Springer, Heidelberg, Germany, Gold Coast, Australia (Dec 9–13, 2001)
6. Camenisch, J., Groß, T.: Efficient attributes for anonymous credentials. In: Ning, P., Syverson, P.F., Jha, S. (eds.) ACM CCS 2008: 15th Conference on Computer and Communications Security. pp. 345–356. ACM Press, Alexandria, Virginia, USA (Oct 27–31, 2008)
7. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann, B. (ed.) Advances in Cryptology – EUROCRYPT 2001. Lecture Notes in Computer Science, vol. 2045, pp. 93–118. Springer, Heidelberg, Germany, Innsbruck, Austria (May 6–10, 2001)
8. Camenisch, J., Michels, M.: Proving in zero-knowledge that a number is the product of two safe primes. In: Stern, J. (ed.) Advances in Cryptology – EUROCRYPT'99. Lecture Notes in Computer Science, vol. 1592, pp. 107–122. Springer, Heidelberg, Germany, Prague, Czech Republic (May 2–6, 1999)
9. Chairattana-Apirom, R., Lysyanskaya, A.: Compact cut-and-choose: Boosting the security of blind signature schemes, compactly. Cryptology ePrint Archive, Report 2022/003 (2022), https://eprint.iacr.org/2022/003
10. Chaum, D.: Blind signatures for untraceable payments. In: Chaum, D., Rivest, R.L., Sherman, A.T. (eds.) Advances in Cryptology – CRYPTO'82. pp. 199–203. Plenum Press, New York, USA, Santa Barbara, CA, USA (1982)
11. Crandall, R., Pomerance, C.B.: Prime numbers: a computational perspective, vol. 182. Springer Science & Business Media (2006)
12. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) Advances in Cryptology – CRYPTO'86. Lecture Notes in Computer Science, vol. 263, pp. 186–194. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 1987)
13. Fischlin, M.: Communication-efficient non-interactive proofs of knowledge with online extractors. In: Shoup, V. (ed.) Advances in Cryptology – CRYPTO 2005. Lecture Notes in Computer Science, vol. 3621, pp. 152–168. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2005)
14. Fischlin, M.: Round-optimal composable blind signatures in the common reference string model. In: Dwork, C. (ed.) Advances in Cryptology – CRYPTO 2006. Lecture Notes in Computer Science, vol. 4117, pp. 60–77. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2006)
15. Fuchsbauer, G., Hanser, C., Slamanig, D.: Practical round-optimal blind signatures in the standard model. In: Gennaro, R., Robshaw, M.J.B. (eds.) Advances in Cryptology – CRYPTO 2015, Part II. Lecture Notes in Computer Science, vol. 9216, pp. 233–253. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2015)
16. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018, Part II. Lecture Notes in Computer Science, vol. 10992, pp. 33–62. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2018)
17. Fuchsbauer, G., Plouviez, A., Seurin, Y.: Blind schnorr signatures and signed ElGamal encryption in the algebraic group model. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology – EUROCRYPT 2020, Part II. Lecture Notes in Computer Science, vol. 12106, pp. 63–95. Springer, Heidelberg, Germany, Zagreb, Croatia (May 10–14, 2020)
18. Garg, S., Gupta, D.: Efficient round optimal blind signatures. In: Nguyen, P.Q., Oswald, E. (eds.) Advances in Cryptology – EUROCRYPT 2014. Lecture Notes

in Computer Science, vol. 8441, pp. 477–495. Springer, Heidelberg, Germany, Copenhagen, Denmark (May 11–15, 2014)
19. Garg, S., Rao, V., Sahai, A., Schröder, D., Unruh, D.: Round optimal blind signatures. In: Rogaway, P. (ed.) Advances in Cryptology – CRYPTO 2011. Lecture Notes in Computer Science, vol. 6841, pp. 630–648. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2011)
20. Ghadafi, E.: Efficient round-optimal blind signatures in the standard model. In: Kiayias, A. (ed.) FC 2017: 21st International Conference on Financial Cryptography and Data Security. Lecture Notes in Computer Science, vol. 10322, pp. 455–473. Springer, Heidelberg, Germany, Sliema, Malta (Apr 3–7, 2017)
21. Goldberg, S., Reyzin, L., Sagga, O., Baldimtsi, F.: Efficient noninteractive certification of RSA moduli and beyond. In: Galbraith, S.D., Moriai, S. (eds.) Advances in Cryptology – ASIACRYPT 2019, Part III. Lecture Notes in Computer Science, vol. 11923, pp. 700–727. Springer, Heidelberg, Germany, Kobe, Japan (Dec 8–12, 2019)
22. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions (extended abstract). In: 25th Annual Symposium on Foundations of Computer Science. pp. 464–479. IEEE Computer Society Press, Singer Island, Florida (Oct 24–26, 1984)
23. Grontas, P., Pagourtzis, A., Zacharakis, A., Zhang, B.: Towards everlasting privacy and efficient coercion resistance in remote electronic voting. In: Zohar, A., Eyal, I., Teague, V., Clark, J., Bracciali, A., Pintore, F., Sala, M. (eds.) FC 2018 Workshops. Lecture Notes in Computer Science, vol. 10958, pp. 210–231. Springer, Heidelberg, Germany, Nieuwpoort, Curaçao (Mar 2, 2019)
24. Guillou, L.C., Quisquater, J.J.: A "paradoxical" indentity-based signature scheme resulting from zero-knowledge. In: Goldwasser, S. (ed.) Advances in Cryptology – CRYPTO'88. Lecture Notes in Computer Science, vol. 403, pp. 216–231. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 21–25, 1990)
25. Hauck, E., Kiltz, E., Loss, J.: A modular treatment of blind signatures from identification schemes. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2019, Part III. Lecture Notes in Computer Science, vol. 11478, pp. 345–375. Springer, Heidelberg, Germany, Darmstadt, Germany (May 19–23, 2019)
26. Hauck, E., Kiltz, E., Loss, J., Nguyen, N.K.: Lattice-based blind signatures, revisited. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology – CRYPTO 2020, Part II. Lecture Notes in Computer Science, vol. 12171, pp. 500–529. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2020)
27. Heilman, E., Baldimtsi, F., Goldberg, S.: Blindly signed contracts: Anonymous on-blockchain and off-blockchain bitcoin transactions. In: Clark, J., Meiklejohn, S., Ryan, P.Y.A., Wallach, D.S., Brenner, M., Rohloff, K. (eds.) FC 2016 Workshops. Lecture Notes in Computer Science, vol. 9604, pp. 43–60. Springer, Heidelberg, Germany, Christ Church, Barbados (Feb 26, 2016)
28. Juels, A., Luby, M., Ostrovsky, R.: Security of blind digital signatures (extended abstract). In: Kaliski Jr., B.S. (ed.) Advances in Cryptology – CRYPTO'97. Lecture Notes in Computer Science, vol. 1294, pp. 150–164. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 1997)
29. Kastner, J., Loss, J., Xu, J.: On pairing-free blind signature schemes in the algebraic group model. In: PKC 2022 (to appear). Lecture Notes in Computer Science, Springer, Heidelberg, Germany (2022)
30. Katz, J., Loss, J., Rosenberg, M.: Boosting the security of blind signature schemes. In: Advances in Cryptology – ASIACRYPT 2021. Lecture Notes in Computer Science, vol. 13093, pp. 468–492. Springer International Publishing, Cham (2021)

31. Okamoto, T.: Provably secure and practical identification schemes and corresponding signature schemes. In: Brickell, E.F. (ed.) Advances in Cryptology – CRYPTO'92. Lecture Notes in Computer Science, vol. 740, pp. 31–53. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 1993)
32. Okamoto, T.: Efficient blind and partially blind signatures without random oracles. In: Halevi, S., Rabin, T. (eds.) TCC 2006: 3rd Theory of Cryptography Conference. Lecture Notes in Computer Science, vol. 3876, pp. 80–99. Springer, Heidelberg, Germany, New York, NY, USA (Mar 4–7, 2006)
33. Okamoto, T., Ohta, K.: Universal electronic cash. In: Feigenbaum, J. (ed.) Advances in Cryptology – CRYPTO'91. Lecture Notes in Computer Science, vol. 576, pp. 324–337. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 11–15, 1992)
34. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) Advances in Cryptology – CRYPTO'91. Lecture Notes in Computer Science, vol. 576, pp. 129–140. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 11–15, 1992)
35. Pointcheval, D.: Strengthened security for blind signatures. In: Nyberg, K. (ed.) Advances in Cryptology – EUROCRYPT'98. Lecture Notes in Computer Science, vol. 1403, pp. 391–405. Springer, Heidelberg, Germany, Espoo, Finland (May 31 – Jun 4, 1998)
36. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. Journal of Cryptology 13(3), 361–396 (Jun 2000)
37. Pollard, J.M.: Monte carlo methods for index computation mod p. Mathematics of computation 32(143), 918–924 (1978)
38. Sahai, A., Waters, B.: How to use indistinguishability obfuscation: deniable encryption, and more. In: Shmoys, D.B. (ed.) 46th Annual ACM Symposium on Theory of Computing. pp. 475–484. ACM Press, New York, NY, USA (May 31 – Jun 3, 2014)
39. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) Advances in Cryptology – EUROCRYPT'97. Lecture Notes in Computer Science, vol. 1233, pp. 256–266. Springer, Heidelberg, Germany, Konstanz, Germany (May 11–15, 1997)
40. Tessaro, S., Zhu, C.: Short pairing-free blind signatures with exponential security. Cryptology ePrint Archive, Report 2022/047 (2022), `https://eprint.iacr.org/2022/047`
41. Wagner, B., Hanzlik, L., Loss, J.: PI-cut-choo! Parallel instance cut and choose for practical blind signatures. Cryptology ePrint Archive, Report 2022/007 (2022), `https://eprint.iacr.org/2022/007`

Supplementary Material

A Standard Computational Assumptions

Definition 8 (RSA assumption). *Let* RSAGen *be an algorithm that on input* 1^n *outputs* (N, p, q, e)*, where* $N = pq$ *for distinct* n*-bit primes* p, q *and* $e \in \mathbb{N}$ *such that* $\gcd(e, \varphi(N)) = 1$.

We say that the RSA *assumption holds relative to* RSAGen *if for all PPT algorithms* $\mathcal{A}$ *the following advantage is negligible:*

$$\Pr\left[x^e = y \mid (N, p, q, e) \leftarrow \mathsf{RSAGen}(1^n), \bar{x} \leftarrow\!\!\$\, \mathbb{Z}_N^*, y := \bar{x}^e, x \leftarrow \mathcal{A}(N, e, y)\right].$$

Definition 9 (CDH assumption). *Let* PGGen *be an algorithm that on input* 1^n *outputs* $(\mathbb{G}, g, p, e)$*, where* $\mathbb{G}$ *is the description of a cyclic group with generator* g *and prime order* p*, and* $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ *is a non-degenerate bilinear map into some target group* $\mathbb{G}_T$*.*

We say that the CDH *assumption holds relative to* PGGen *if for all PPT algorithms* $\mathcal{A}$ *the following advantage is negligible:*

$$\Pr\left[z = xy \mid (\mathbb{G}, g, p, e) \leftarrow \mathsf{PGGen}(1^n), a, b \leftarrow_{\$} \mathbb{Z}_p, g^z \leftarrow \mathcal{A}(\mathbb{G}, g, p, e, g^x, g^y)\right].$$

B From Semi-Honest to Malicious Blindness

Here we show how to transform any blind signature scheme with semi-honest signer blindness into a scheme that satisfies malicious signer blindness. In summary, we show the following lemma.

Lemma 1 (Informal). *Let* BS *be a blind signature scheme that satisfies semi-honest signer blindness. Then, using a non-interactive zero-knowledge proof-of-knowledge,* BS *can be transformed into a blind signature scheme* BS' *that satisfies malicious signer blindness. Furthermore, the schemes* BS *and* BS' *are identical except for public keys* pk*. Finally, for any* $\ell : \mathbb{N} \to \mathbb{N}$*, it holds that if* BS *satisfies* ℓ-OMUF*, then* BS' *satisfies* ℓ-OMUF *and the security proofs of this transformation are tight.*

B.1 Non-Interactive Proof Systems

Before we describe our transformation, we recall the definition of non-interactive proof systems. For simplicity of exposition, we focus on online-extractable proof systems in the random oracle model as defined in [13]. However, we note that different notions of non-interactive proofs are also applicable here.

Let $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ be a binary relation. We define the associated language $\mathcal{L}_{\mathcal{R}} \subseteq \{0,1\}^*$ via

$$x \in \mathcal{L}_{\mathcal{R}} \iff (x, w) \in \mathcal{R} \text{ for some } w \in \{0,1\}^*,$$

for all $x \in \{0,1\}^*$. Here, we call x the statement and w the witness. In the following, we focus on **NP** relations, which means that $\mathcal{R}$ is efficiently decidable and the length of the witness w is bounded by a polynomial in the length of the statement x. Relations can implicitly depend on the security parameter, with the restriction that the length of statments is polynomially bounded in the security parameter.

Definition 10 (Non-interactive Proofs). *Let* $\mathcal{R}$ *be an* **NP** *relation and* H *be a random oracle. A non-interactive zero-knowledge proof-of-knowledge (NIZKPOK) for* $\mathcal{R}$ *is a tuple* $\mathsf{PS} = (\mathsf{PProve}, \mathsf{PVer})$ *such that*

- $\mathsf{PProve}^{\mathsf{H}}(x, w)$ *takes as input a statement* x *and a witness* w *and outputs a proof* π*.*

– $\mathsf{PVer}^{\mathsf{H}}(x, \pi)$ *is deterministic, takes as input a statement* x *and a proof* π *and outputs a bit* $b \in \{0, 1\}$.

Further, the following completeness and security properties should hold:

– ***Completeness.*** *For all* $(x, w) \in \mathcal{R}$ *we have*

$$\Pr\left[\mathsf{PVer}^{\mathsf{H}}(x, \pi) = 1 \mid \pi \leftarrow \mathsf{PProve}^{\mathsf{H}}(x, w)\right] = 1.$$

– ***Zero-Knowledge.*** *There exists a PPT algorithm* Sim *such that for any PPT algorithm* $\mathcal{D}$ *the following advantage is negligible:*

$$\begin{aligned} &|\Pr\left[\mathcal{D}^{\mathsf{H}}(St, \pi) = 1 \middle| (St, x, w) \leftarrow \mathcal{D}^{\mathsf{H}}(1^n), \pi \leftarrow \mathsf{PProve}^{\mathsf{H}}(x, w)\right] \\ &- \Pr\left[\mathcal{D}^{\mathsf{H}_1}(St, \pi) = 1 \middle| \begin{array}{l}(St_{\mathsf{Sim}}, \mathsf{H}_0) \leftarrow \mathsf{Sim}(1^n), \\ (St, x, w) \leftarrow \mathcal{D}^{\mathsf{H}_0}(1^n), (\mathsf{H}_1, \pi) \leftarrow \mathsf{Sim}(St_{\mathsf{Sim}}, x)\end{array}\right]|. \end{aligned}$$

– ***Proof-of-Knowledge.*** *There exists a PPT algorithm* Ext *such that for any algorithm* $\mathcal{A}$ *the following advantage is negligible:*

$$\Pr\left[(x, w) \notin \mathcal{R} \wedge \mathsf{PVer}^{\mathsf{H}}(x, \pi) = 1 \middle| (x, \pi) \leftarrow \mathcal{A}^{\mathsf{H}}(1^n), w \leftarrow \mathsf{Ext}(x, \pi, \mathcal{Q})\right],$$

where $\mathcal{Q}$ *denotes the list of queries of* $\mathcal{A}$ *to oracle* H *and the respective answers.*

B.2 Construction

We describe how we turn any blind signature scheme satisfying semi-honest signer blindness into a blind signature scheme satisfying malicious signer blindness. Our transformation preserves one-more unforgeability.

Let $\mathsf{BS} = (\mathsf{Gen}, \mathsf{S}, \mathsf{U}, \mathsf{Ver})$ be a blind signature scheme. Consider the relation of public keys and random coins

$$\mathcal{R} = \{(\mathsf{pk}, \rho) \mid \exists \mathsf{sk} : (\mathsf{pk}, \mathsf{sk}) = \mathsf{Gen}(1^n; \rho)\},$$

where pk is the statement and ρ is the witness. Further, let $\mathsf{PS} = (\mathsf{PProve}, \mathsf{PVer})$ be a NIZKPOK for $\mathcal{R}$ using random oracle H.

We transform BS into a new blind signature scheme $\mathsf{BS}' = (\mathsf{Gen}', \mathsf{S}, \mathsf{U}', \mathsf{Ver})$, where algorithms S and Ver stay the same. Algorithm Gen' is as follows:

1. Sample random coins ρ for algorithm Gen.
2. Generate $(\mathsf{pk}_0, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^n; \rho)$.
3. Compute a proof $\pi \leftarrow \mathsf{PProve}^{\mathsf{H}}(\mathsf{pk}_0, \rho)$ using pk_0 as the statement and ρ as the witness.
4. Return the public key $\mathsf{pk} := (\mathsf{pk}_0, \pi)$ and the secret key sk.

Further, algorithm U' is as algorithm U, but additionally checks the correctness of the public key pk first. That is, U' parses $\mathsf{pk} = (\mathsf{pk}_0, \pi)$ and runs $b := \mathsf{PVer}^{\mathsf{H}}(\mathsf{pk}_0, \pi)$. Then, if $b = 0$, it aborts the interaction. If $b = 1$, it behaves as U does.

B.3 Analysis

We now show that one-more unforgeability is preserved and we achieve malicious signer blindness.

Lemma 2. *Let* BS *be a blind signature scheme and* PS *be a NIZKPOK for the relation* $\mathcal{R}$ *using random oracle* H*. Then,* BS' *satisfies malicious signer blindness assuming that* BS *satisfies semi-honest signer blindness.*

Concretely, let t_{Ext} *denote the running time of the extractor algorithm* Ext *given by the proof-of-knowledge property of* PS*. Then, for any adversary against the malicious signer blindness of* BS' *with advantage* ϵ *and running time* t*, there exist adversaries against the proof-of-knowledge property of* PS *and against the semi-honest signer blindness of* BS *with running time* $t_{\mathsf{PS}} = t, t_{\mathsf{BS}} \leq t + t_{\mathsf{Ext}}$ *and advantage* $\epsilon_{\mathsf{PS}}, \epsilon_{\mathsf{BS}}$*, respectively, such that* $\epsilon \leq \epsilon_{\mathsf{BS}} + \epsilon_{\mathsf{PS}}$*.*

Proof. Let $\mathcal{A}$ be an adversary against malicious signer blindness of BS'. We denote its advantage by ϵ. First, we can assume that $\mathcal{A}$ outputs a key $\mathsf{pk} = (\mathsf{pk}_0, \pi)$ such that $\mathsf{PVer}^{\mathsf{H}}(\mathsf{pk}_0, \pi) = 1$. This is because otherwise, user oracles of scheme BS' abort and leak no information about the message that is used.

With this assumption in mind, we build a reduction $\mathcal{B}$ against semi-honest signer blindness of BS. The reduction uses $\mathcal{A}$ as a subroutine. It also makes use of the extractor Ext that exists due to the proof-of-knowledge property of PS. Reduction $\mathcal{B}$ is as follows:

- $\mathcal{B}$ simulates the random oracle H for $\mathcal{A}$, keeping track of the list $\mathcal{Q}$ of random oracle queries and answers.
- $\mathcal{B}$ obtains a key pk and messages $\mathsf{m}_0, \mathsf{m}_1$ from $\mathcal{A}$. It parses $\mathsf{pk} = (\mathsf{pk}_0, \pi)$ and runs $\rho \leftarrow \mathsf{Ext}(x, \pi, \mathcal{Q})$. If $(\mathsf{pk}_0, \rho) \notin \mathcal{R}$, $\mathcal{B}$ sets $\mathsf{bad} = 1$ and simulates random oracles $\mathrm{O}'_0, \mathrm{O}'_1$ to adversary $\mathcal{A}$ by aborting each interaction. Later it returns $\sigma_0 = \ \perp, \sigma_1 = \ \perp$ to $\mathcal{A}$. Otherwise, if $(\mathsf{pk}_0, \rho) \in \mathcal{R}$, $\mathcal{B}$ outputs the random coins ρ and the messages $\mathsf{m}_0, \mathsf{m}_1$ to its blindness game. Note that this implies that its blindness game will use the public key pk_0.
- $\mathcal{B}$ gets access to oracles $\mathrm{O}_0, \mathrm{O}_1$. It provides oracles $\mathrm{O}'_0, \mathrm{O}'_1$ to adversary $\mathcal{A}$. By our assumption, we have $\mathsf{PVer}^{\mathsf{H}}(\mathsf{pk}_0, \pi) = 1$. Reduction $\mathcal{B}$ simulates O'_0 as O'_0 and O'_1 as O'_1.
- It runs $\mathcal{A}$ with one-time access to oracles $\mathrm{O}'_0, \mathrm{O}'_1$. Then, it obtains signatures σ_0, σ_1 from its blindness game and forwards them to $\mathcal{A}$.
- When $\mathcal{A}$ outputs a bit b', $\mathcal{B}$ forwards this bit to its blindness game.

It is clear that the running time of $\mathcal{B}$ is dominated by the running time of $\mathcal{A}$ and the running time of Ext. Further, assuming that bad is never set to 1, $\mathcal{B}$ perfectly simulates the malicious signer blindness game $\mathbf{BLIND}_{b,\mathsf{BS}'}$ for $\mathcal{A}$ if it runs in semi-honest signer blindness game $\mathbf{BLIND}_{b,\mathsf{BS}}$. Also, note that the probability of $\mathsf{bad} = 1$ is bounded by the knowledge error of PS, which is ϵ_{PS} by assumption. Further, the probability of $\mathsf{bad} = 1$ is independent of the bit b. To conclude the

proof, we define the event $\mathsf{W}_b := \left(\mathbf{BLIND}^{\mathcal{A}}_{b,\mathsf{BS}'}(n) \Rightarrow 1\right)$. We obtain

$$\begin{aligned}
\epsilon = \quad & |\Pr[\mathsf{W}_0] - \Pr[\mathsf{W}_1]| \\
\leq \quad & |\Pr[\mathsf{W}_0 \mid \mathsf{bad}=0] \cdot \Pr[\mathsf{bad}=0] + \Pr[\mathsf{W}_0 \mid \mathsf{bad}=1] \cdot \Pr[\mathsf{bad}=1] \\
& -\Pr[\mathsf{W}_1 \mid \mathsf{bad}=0] \cdot \Pr[\mathsf{bad}=0] + \Pr[\mathsf{W}_1 \Rightarrow 1 \mid \mathsf{bad}=1] \cdot \Pr[\mathsf{bad}=1]| \\
\leq \quad & |\Pr[\mathsf{W}_0 \mid \mathsf{bad}=0] - \Pr[\mathsf{W}_1 \mid \mathsf{bad}=0]| \cdot \Pr[\mathsf{bad}=0] \\
& +|\Pr[\mathsf{W}_0 \mid \mathsf{bad}=1] - \Pr[\mathsf{W}_1 \mid \mathsf{bad}=1]| \cdot \Pr[\mathsf{bad}=1] \\
\leq \quad & |\Pr[\mathsf{W}_0 \mid \mathsf{bad}=0] - \Pr[\mathsf{W}_1 \mid \mathsf{bad}=0]| + \Pr[\mathsf{bad}=1] \leq \epsilon_{\mathsf{BS}} + \epsilon_{\mathsf{PS}}.
\end{aligned}$$

□

Lemma 3. *Let* BS *be a blind signature scheme,* PS *be a NIZKPOK for the relation* $\mathcal{R}$ *using random oracle* H*, and* $\ell : \mathbb{N} \to \mathbb{N}$ *be a function. Then,* BS′ *satisfies* ℓ*-one-more unforgeability assuming that* BS *satisfies* ℓ*-one-more unforgeability.*

Concretely, let t_{Sim} *denote the running time of the simulator algorithm* Sim *given by the zero-knowledge property of* PS*. Then, for any adversary against the* ℓ*-one-more unforgeabilit of* BS′ *with advantage* ϵ *and running time* t*, there exist adversaries against the zero-knowledge property of* PS *and against the* ℓ*-one-more unforgeabilit of* BS *with running time* $t_{\mathsf{PS}} = t, t_{\mathsf{BS}} \leq t + t_{\mathsf{Sim}}$ *and advantage* $\epsilon_{\mathsf{PS}}, \epsilon_{\mathsf{BS}}$*, respectively, such that* $\epsilon \leq \epsilon_{\mathsf{BS}} + \epsilon_{\mathsf{PS}}$*.*

Proof. Let $\mathcal{A}$ be an adversary against the ℓ-OMUF security of BS′. We show the statement via a reduction from the ℓ-OMUF security of BS.

Game $\mathbf{G}_0$: We start with game $\mathbf{G}_0 := \ell\text{-}\mathbf{OMUF}^{\mathcal{A}}_{\mathsf{BS}'}$, which is the one-more unforgeability game. First, a key pair $(\mathsf{pk}, \mathsf{sk})$ is sampled by the game. Recall that pk is of the form $\mathsf{pk} = (\mathsf{pk}_0, \pi)$, where $\pi \leftarrow \mathsf{PProve}^{\mathsf{H}}(\mathsf{pk}_0, \rho)$ and ρ are the random coins used to generate pk_0. Then, $\mathcal{A}$ is executed with input pk and oracle access to random oracle H and a signer oracle O′. In the end, $\mathcal{A}$ outputs pairs of signatures and messages and the game outputs 1 if these are all valid, the messages are distinct and there are more such pairs than the number of completed interactions with oracle O′.

Game $\mathbf{G}_1$: We change the way the proof π contained in pk is generated. Namely, we use the simulator Sim that exists by the zero-knowledge property of PS to generate π. Note that this simulator may program the random oracle H. Clearly, we can bound the difference between $\mathbf{G}_0$ and $\mathbf{G}_1$ by the advantage ϵ_{PS} of a reduction against the zero-knowledge property of PS. Thus, we have

$$|\Pr[\mathbf{G}_0 \Rightarrow 1] - \Pr[\mathbf{G}_1 \Rightarrow 1]| \leq \epsilon_{\mathsf{PS}}.$$

We will now bound the probability that $\mathbf{G}_1$ outputs 1 by a reduction $\mathcal{B}$ from the ℓ-OMUF security of BS. We denote the advantage of $\mathcal{B}$ by ϵ_{BS}. The reduction is as follows:

- $\mathcal{B}$ gets as input a public key pk_0 for scheme BS. Also, $\mathcal{B}$ gets oracle access to a signer oracle O for scheme BS. It generates a proof π for the statement pk_0 using the zero-knowledge simulator Sim and defines $\mathsf{pk} := (\mathsf{pk}_0, \pi)$.

- $\mathcal{B}$ runs algorithm $\mathcal{A}$ on input pk by providing random oracle H and a signer oracle O', which is simulated using O.
- $\mathcal{B}$ forwards the outputs of $\mathcal{A}$ to its own game.

It is easy to see that the reduction perfectly simulates $\mathbf{G}_1$ for $\mathcal{A}$. Also, as verification for BS is the same as for BS', any valid forgery of $\mathcal{A}$ leads to a valid forgery of $\mathcal{B}$. Thus, we have

$$\Pr\left[\mathbf{G}_1 \Rightarrow 1\right] \leq \epsilon_{\mathsf{BS}}.$$

□

B.4 Applicability to Our Schemes

Let us discuss how to apply the transformation that we presented in this section to our schemes. We focus on the schemes resulting from the generic construction in Section 3 and the concrete scheme from RSA.

Using Generic Proofs. We can use a generic zero-knowledge proof-of-knowledge for **NP** to apply the transformation to any blind signature scheme that satisfies semi-honest signer blindness. While this is an inefficient solution in general, we argue that it is acceptable here. First, such proofs are allowed to use random oracles in our setting. Second, the proof only needs to be generated once, and verified once by each user. Therefore, this will only induce a one-time overhead, which is independent of the complexity of the actual signing protocol.

More Efficient Solutions. Taking into account the concrete structure of the schemes we construct, more efficient solutions are possible. For example, the relation between a public and a secret key (which is part of the random coins used for key generation) in our protocols is given by a linear function. Therefore, simple Schnorr-style Fiat-Shamir [12] proofs can be used. In an RSA-based setting, one also needs to prove knowledge of the coins needed for the generation of parameters.

As an example, consider our RSA-based scheme from the OGQ linear function family, as presented in Supplementary Material Section H. Here, the random coins used for key generation are given by $p, q, a, \lambda, \mathsf{sk}'$ and the random coins used to generate ck. First, one can show that N is the product of two primes p, q using techniques from [21] or [8]. Here, one needs resort to the quadratic residuosity assumption [21] or the discrete logarithm assumption [8]. Also, using [21], one can show that $\gcd(\varphi(N), \lambda) = 1$. We can then turn this into a proof-of-knowledge by running a non-interactive version of the Fiat-Shamir identification scheme [12] on random public keys (e.g. sampled via the random oracle). This proves knowledge of the factorization of N, i.e. of p and q. Similar techniques can be used to prove knowledge of the random coins used to generate ck.

C Blind Signature Schemes from Linear Function Families

Here, we give the concrete security theorem from [25]. It states that for a collision resistant linear function family with pseudo torsion-free element in the kernel,

the linear blind signature scheme $\mathsf{BS}[\mathsf{LF}]$ is secure, as long as only logarithmically many signatures are issued.

Definition 11 (Collision Resistance). *A linear function family* LF *is collision resistant if for any adversary* $\mathcal{A}$*, the following probability is negligible:*

$$\Pr\left[\mathsf{F}(x) = \mathsf{F}(x') \wedge x \neq x' | (x, x') \leftarrow \mathcal{A}(\mathsf{par})\right]$$

Theorem 5 ([25]). *Let* LF *be a linear function family and* $\mathsf{H} : \{0,1\}^* \to \mathcal{S}$ *be a random oracle. If* LF *is collision resistant and has a pseudo torsion-free element in the kernel, then* $\mathsf{BS}[\mathsf{LF}]$ *is* ℓ *for* $\ell = \mathcal{O}(\log n)$*.*

Concretely, for any adversary $\mathcal{A}$ *that has success probability* ϵ *in the* ℓ*-one-more unforgeability game against* $\mathsf{BS}[\mathsf{LF}]$ *and runs in time* t*, initiates at most* p *protocol execution, and makes at most* Q_{H} *queries to* H*, there exists an algorithm against the collision resistance of* LF *with running time* $t' = 2t$ *and advantage* ϵ'*, where*

$$\epsilon' = \Omega\left(\left(\frac{\epsilon}{2} - \frac{(Q \cdot (p-\ell))^{\ell+1}}{2^{2n}}\right)^3 \frac{1}{Q^2 \ell^3}\right)$$

and $Q = Q_{\mathsf{H}} + \ell + 1$*.*

Next, we show semi-honest signer blindness of $\mathsf{BS}[\mathsf{LF}]$. We note that [25] only claim honest signer blindness, but their proof carries over to semi-honest signer blindness.

Theorem 6. *Let* LF *be a linear function family and* $\mathsf{H} : \{0,1\}^* \to \mathcal{S}$*. Then* $\mathsf{BS}[\mathsf{LF}]$ *satisfies semi-honest signer blindness.*

Concretely, every adversary against the semi-honest signer blindness of $\mathsf{BS}[\mathsf{LF}]$ *that queries* H *at most* Q_{H} *times has advantage at most* $4Q_{\mathsf{H}}/|\mathcal{R}|$*.*

Proof. We follow the proof given in [25]. Consider an adversary $\mathcal{A}$ against the semi-honest signer blindness of $\mathsf{BS}[\mathsf{LF}]$. In the game $\mathbf{G}_b := \mathbf{BLIND}_{b,\mathsf{BS}[\mathsf{LF}]}$ for bit $b \in \{0,1\}$, $\mathcal{A}$ first outputs random coins ρ, and two messages $\mathsf{m}_0, \mathsf{m}_1$. The random coins define a public key pk and a secret key sk. We write $\mathsf{pk} = \mathsf{F}(\mathsf{sk})$, keeping in mind that pk also contains parameters par specifying F. Note that these parameters are the output of the key generation algorithm and therefore well-formed. Then, $\mathcal{A}$ can interact with two user oracles $\mathrm{O}_0, \mathrm{O}_1$. We denote the randomness that is used by these oracles by $(\alpha_0, \beta_0), (\alpha_1, \beta_1)$, respectively. Further, we denote the transcripts that $\mathcal{A}$ learns during this interaction by $T_0 = (R_0, c_0, s_0)$ and $T_1 = (R_1, c_1, s_1)$, respectively. After the interaction, $\mathcal{A}$ obtains signatures σ_0 for m_0 and σ_1 for m_1.

In a first step of our proof, we rule out the bad event that $\mathcal{A}$ queries $\mathsf{H}(\mathsf{m}_b, R'_0)$ or $\mathsf{H}(\mathsf{m}_{1-b}, R'_1)$, where $R'_0 = R_0 + \mathsf{F}(\alpha_0) + \beta_0 \cdot \mathsf{pk}$ and $R'_1 = R_1 + \mathsf{F}(\alpha_1) + \beta_1 \cdot \mathsf{pk}$ before $\mathcal{A}$ obtains the signatures σ_0, σ_1. As F is smooth and α_0, α_1 are chosen uniformly at random from $\mathcal{D}$, this bad event occurs with probability at most $2Q_{\mathsf{H}}/|\mathcal{R}|$. We call the resulting game $\mathbf{G}'_b$.

Now, we argue that the view of $\mathcal{A}$ in game $\mathbf{G}'_0$ is the same as the view of $\mathcal{A}$ in game $\mathbf{G}'_1$. Then, the claim follows from

$$\begin{aligned}|\Pr[\mathbf{G}_0 \Rightarrow 1] - \Pr[\mathbf{G}_1 \Rightarrow 1]| \leq &\left|\Pr[\mathbf{G}_0 \Rightarrow 1] - \Pr\left[\mathbf{G}'_0 \Rightarrow 1\right]\right| \\ &+ \left|\Pr\left[\mathbf{G}'_0 \Rightarrow 1\right] - \Pr\left[\mathbf{G}'_1 \Rightarrow 1\right]\right| \\ &+ \left|\Pr\left[\mathbf{G}'_1 \Rightarrow 1\right] - \Pr[\mathbf{G}_1 \Rightarrow 1]\right| \leq \frac{4Q_{\mathsf{H}}}{|\mathcal{R}|}.\end{aligned}$$

We argue that for all combinations of $i \in \{0,1\}$ and $j \in \{0,1\}$, there are values $(\alpha_{i,j}, \beta_{i,j})$ that explain that m_j was used in oracle O_i and have the same distribution before $\mathcal{A}$ obtains the signatures. To this end, we look at the experiment before $\mathcal{A}$ obtains signatures σ_0, σ_1. Here, define the values

$$\alpha_{i,j} := s'_j - s_i - \Psi(\mathsf{pk}, -c'_j, c_i), \quad \beta_{i,j} = c_i - c'_j$$

for $i, j \in \{0,1\}$. We see that for all $i, j \in \{0,1\}$ the value $\beta_{i,j}$ is distributed uniformly conditioned on the view of $\mathcal{A}$ before learning the signatures. This is because the value $c'_j = \mathsf{H}(\mathsf{m}_j, R'_{j\oplus b})$ is uniform as we ruled out that $\mathcal{A}$ queries the random oracle at that position. Also, the distribution of $\alpha_{i,j} = s_j + \alpha_j + \Psi(\mathsf{pk}, -c'_j, c_{j\oplus b}) - s_i - \Psi(\mathsf{pk}, -c'_j, c_i)$ is uniform, due to the uniform choice of α_j.

Further, the values $(\alpha_{i,j}, \beta_{i,j})$ satisfy $c'_j = \mathsf{H}(\mathsf{m}_j, R_i + \beta_{i,j}\mathsf{pk} + \mathsf{F}(\alpha_{i,j}))$ for all $i, j \in \{0,1\}$, which can be verified by an easy calculation using $\mathsf{pk} = \mathsf{F}(\mathsf{sk})$, see [25].

Thus, the view of $\mathcal{A}$ before and after it obtains the signatures σ_0, σ_1 is independent of the bit b, which finishes the proof. □

D Randomness Homomorphic Commitments

Here, we show how randomness homomorphic commitment schemes can be obtained from standard assumptions, such as RSA and DLOG. Before we give concrete schemes from these assumptions, we introduce a generic way of obtaining such commitment schemes from linear function families.

To this end, let $\mathsf{LF} = (\mathsf{PGen}, \mathsf{F}, \Psi)$ be a linear function family. For $\mathsf{par} \leftarrow \mathsf{PGen}(1^n)$, a commitment key ck is a random element $\mathsf{ck} \leftarrow\!\!\$\, \mathcal{R}$ in the range of the linear function. We assume that ck implicitly contains par. We the define a commitment for element $x \in \mathcal{S}$ and randomness $r \in \mathcal{D}$, and a translation of a commitment $\mu \in \mathcal{R}$ by r as follows:

$$\mathsf{Com}(\mathsf{ck}, x; r) := \mathsf{F}(r) - x \cdot \mathsf{ck}, \quad \mathsf{Translate}(\mathsf{ck}, \mu, r) := \mu + \mathsf{F}(r).$$

By smoothness of the linear function F, we see that this is perfectly hiding. Completeness of translation follows by linearity of F. In most cases, the binding property follows from the preimage resistance of LF and[7]

$$\mathsf{Com}(\mathsf{ck}, x_0; r_0) = \mathsf{Com}(\mathsf{ck}, x_1; r_1) \Rightarrow \mathsf{F}(r_0 - r_1) = (x_0 - x_1) \cdot \mathsf{ck}.$$

[7] This corresponds to special soundness if the linear identification scheme derived from LF. Note that this is always given if the set of scalars is a field. In other cases, one may still be able to show special soundness, e.g. using Shamir's trick.

D.1 Randomness Homomorphic Commitment from RSA

To instantiate the randomness homomorphic commitment scheme, we show that a folklore commitment scheme based on the RSA assumption is randomness homomorphic. From another point of view, this scheme can be obtained from the Guillou-Quisquater identification scheme [24]. The commitment key ck contains public parameters (N, e) such that $N = pq$ for two distinct n-bit primes, e is prime and $\gcd(e, \varphi(N)) = 1$ as well as an element $y := x^e \bmod N$, where $x \leftarrow\!\!\$\, \mathbb{Z}_N^*$. Commitment and translation algorithms for $\mathsf{m} \in \mathbb{Z}_e, r \in \mathbb{Z}_N^*, \mu \in \mathbb{Z}_N^*$ are defined as follows:

$$\mathsf{Com}(\mathsf{ck}, \mathsf{m}; r) := r^e y^{\mathsf{m}} \bmod N$$
$$\mathsf{Translate}(\mathsf{ck}, \mu, r) := r^e \mu \bmod N.$$

It is easy to observe that translation is complete and the commitment is perfectly hiding. To see that it is computationally binding, note that given two pairs $(\mathsf{m}_0, r_0), (\mathsf{m}_1, r_1) \in \mathbb{Z}_e \times \mathbb{Z}_N^*$ with $\mathsf{m}_0 \neq \mathsf{m}_1$ and $\mathsf{Com}(\mathsf{ck}, \mathsf{m}_0; r_0) = \mathsf{Com}(\mathsf{ck}, \mathsf{m}_1; r_1)$ we have

$$r_0^e y^{\mathsf{m}_0} \equiv r_1^e y^{\mathsf{m}_1} \pmod N.$$

Without loss of generality we have $\mathsf{m}_0 > \mathsf{m}_1$ and as e is prime we have $\gcd(e, \mathsf{m}_0 - \mathsf{m}_1) = 1$. Thus, we can apply Shamir's trick to

$$(r_0^{-1} r_1)^e \equiv y^{\mathsf{m}_0 - \mathsf{m}_1} \pmod N$$

and derive an e^{th} root of y.

D.2 Randomness Homomorphic Commitment from DLOG

We also show that the standard Pedersen commitment scheme [34] is randomness homomorphic. Recall that in this scheme, the commitment key is a pair of group elements g, h, where $(\mathbb{G}, g, p) \leftarrow \mathsf{GGen}(1^n)$. Commitment and translation algorithms for $\mathsf{m} \in \mathbb{Z}_p, r \in \mathbb{Z}_p, \mu \in \mathbb{G}$ are defined as follows:

$$\mathsf{Com}(\mathsf{ck}, \mathsf{m}; r) := g^r h^{\mathsf{m}}$$
$$\mathsf{Translate}(\mathsf{ck}, \mu, r) := g^r \mu.$$

It is well-known (and easy to see) that the scheme is perfectly hiding and computationally binding under the DLOG assumption relative to GGen. Also, completeness of translation is easy to see.

E Puncturable Pseudorandom Function

We instantiate the puncturable pseudorandom function PRF using the classical GGM construction [22]. As our framework is defined in the random oracle model, we also instantiate the GGM construction using random oracles. The construction is as follows. Let $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{2n}$ be a random oracle. For simplicity, we

write $\mathsf{H}(x) = (\mathsf{H}_0(x), \mathsf{H}_1(x))$ for any x to separate the output of H into two n-bit strings. Keys are random strings of length n and for $\ell \in \mathbb{N}, x \in \{0,1\}^\ell, k \in \{0,1\}^n$ we define

$$\mathsf{GGM}_{0,k}() := k,\ \mathsf{GGM}_{\ell,k}(b \,\|\, x) := \mathsf{GGM}_{\ell-1,\mathsf{H}_b(k)}(x)$$

Then the evaluation of the pseudorandom function with key $k \in \{0,1\}^n$ on input $x \in \{0,1\}^{d(n)}$ is $\mathsf{PRF.Eval}(k,x) := \mathsf{GGM}_{d(n),k}(x)$. We also define an algorithm $\mathsf{Puncture}_\ell(k, X)$ to puncture keys at a set of points $\emptyset \neq X \subseteq \{0,1\}^\ell$ as follows: We set $\mathsf{Puncture}_0(k, X) := \emptyset$ and

- Set $k_X := \emptyset$ and $(k_0, k_1) := \mathsf{H}(k)$.
- Define sets $X_b := \{x \mid (x_1, x) \in X \wedge x_1 = b\}$ for $b \in \{0,1\}$.
- If $X_0 = \emptyset$, set $k_X := k_X \cup \{k_0\}$. Else set $k_X := k_X \cup \{\mathsf{Puncture}_{\ell-1}(k_0, X_0)\}$.
- If $X_1 = \emptyset$, set $k_X := k_X \cup \{k_1\}$. Else set $k_X := k_X \cup \{\mathsf{Puncture}_{\ell-1}(k_1, X_1)\}$.
- Return k_X.

Note that this algorithm always terminates. We set $\mathsf{PRF.Puncture}(k, X) := \mathsf{Puncture}_{d(n)}(k, X)$. Also, note that punctured a punctured key contains all information needed to evaluate the pseudorandom function at inputs that are not in X. Using a proof by induction over $d(n)$ and $|X|$, one can easily show that the number of elements in k_X is at most $(d(n) - 1)|X| + 1$.

It remains to show pseudorandomness on punctured points.

Lemma 4. *Let $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{2n}$ be a random oracle and consider the puncturable pseudorandom function PRF as defined above. Let $\mathcal{A}$ be a PPT algorithm that makes at most Q queries to H. Then the advantage of $\mathcal{A}$ in the pseudorandomness game for PRF is at most*

$$\frac{(2d(n) - 1)Q|X|}{2^n}$$

where X is the set of points that $\mathcal{A}$ outputs and $d(n)$ is the input length.

Proof. A simple hybrid argument shows that we only have to argue that we have pseudorandomness for keys which are punctured at one point. Then we can show the claim by induction over the input length $d = d(n)$. In particular, we show that for any PPT algorithm making at most Q random oracle queries the advantage can be bounded by $(2d - 1)Q/2^n$. We start with the case of $d = 1$. Let $\mathcal{A}$ be a PPT algorithm and assume it outputs $X \subseteq \{0,1\}, |X| = 1$. Let $k \leftarrow\!\!\$\, \{0,1\}^n$ be a random key. Let $X = \{x\}$. Conditioned on $k_X = \{\mathsf{H}_{1-x}(k)\}$ the value $\mathsf{PRF.Eval}(k, x) = \mathsf{H}_x(k)$ is uniformly random unless $\mathcal{A}$ queries $\mathsf{H}(k)$. As k is sampled uniformly at random and $\mathcal{A}$ can only make a polynomial number of random oracle queries, a union bound shows that the probability that this happens is negligible. In more detail the distinguishing advantage can be upper bounded by $Q/2^n$. Now consider $d > 1$, let $k \leftarrow\!\!\$\, \{0,1\}^n$ be a random key and let $X = \{x\}, x \in \{0,1\}^d$ be $\mathcal{A}$'s initial output. Let k_X, r be the values that $\mathcal{A}$ gets as input after outputting X. Write $x = x_1 \,\|\, \bar{x}$ for $x_1 \in \{0,1\}, \bar{x} \in \{0,1\}^{d-1}$.

We show indistinguishability via a sequence of four games. In the first game, we let k_X be the honestly punctured key $k_X = \{s_{1-x_1} = \mathsf{H}_{1-x_1}(k), k_{\{\bar{x}\}}\}$ and $r = \mathsf{Eval}(k, x)$ be the real evaluation at input x. In the second game, we set $s_{1-x_1} \leftarrow\!\!\$\, \{0,1\}^n$. Note that similarly to the argument for $d = 1$, the adversary $\mathcal{A}$ can only distinguish between these two games, if it queries $\mathsf{H}(k)$, which happens with probability at most $Q/2^n$. In the third game, we sample $r \leftarrow\!\!\$\, \{0,1\}^n$. Note that any distinguisher between the second and the third game can be turned into a distinguisher for input length $d-1$ with the same advantage by a straight forward reduction. Hence, using the induction hypothesis, the advantage of $\mathcal{A}$ in distinguishing the second and the third game can be upper bounded by $(2(d-1)-1)Q/2^n$. Finally, we undo the change we did in the second game. That is, we set $s_{1-x_1} = \mathsf{H}_{1-x_1}(k)$. Again, the advantage of distinguishing between the third and fourth game is at most $Q/2^n$. In total, we obtain that the advantage of $\mathcal{A}$ in distinguishing between the real value of the pseudorandom function at input x and a random string is at most $(2d-1)Q/2^n$. □

F Ommitted Chernoff Bound

Lemma 5. *For a sum X of independent $\{0,1\}$-random variables and any $s > \mathbb{E}[X]$ it holds that*

$$\Pr[X \geq s] \leq \exp(3\mathbb{E}[X] - s).$$

Proof. The proof is similar to [30]. Recall the standard Chernoff bound for all $\delta > 0$:

$$\Pr[X \geq (1+\delta) \cdot \mathbb{E}[X]] \leq \exp\left(-\mathbb{E}[X]\frac{\delta^2}{2+\delta}\right).$$

Using $x^2 > (x+2)(x-2)$ for all $x \geq 0$ we obtain

$$\begin{aligned}\Pr[X \geq s] &= \Pr\left[X \geq \left(1 + \left(\frac{s}{\mathbb{E}[X]} - 1\right)\right) \cdot \mathbb{E}[X]\right] \\ &\leq \exp\left(-\mathbb{E}[X]\frac{(s/\mathbb{E}[X]-1)^2}{2+(s/\mathbb{E}[X]-1)}\right) \\ &\leq \exp\left(-\mathbb{E}[X]\left(\frac{s}{\mathbb{E}[X]} - 3\right)\right) = \exp(3\mathbb{E}[X] - s).\end{aligned}$$

□

G Omitted Analysis of Our Generic Construction

G.1 Blindness

Proof (of Theorem 1). We note that we prove the theorem using the notion of malicious signer blindness. However, the proof also applies for both honest signer

blindness and semi-honest signer blindness, where the only change is the way the final reduction passes keys between its own game and the adversary.

Let $\mathsf{BS} := \mathsf{CCCBS}[\mathsf{LF}]$ and $\mathcal{A}$ be an algorithm with blindness advantage ϵ. That is,

$$\epsilon := \left| \Pr\left[\mathbf{BLIND}^{\mathcal{A}}_{0,\mathsf{BS}}(n) \Rightarrow 1\right] - \Pr\left[\mathbf{BLIND}^{\mathcal{A}}_{1,\mathsf{BS}}(n) \Rightarrow 1\right] \right|.$$

We will bound this advantage via a sequence of games, changing how the oracles $\mathrm{O}_0, \mathrm{O}_1$ behave. To be precise, we present games $\mathbf{G}_{i,b}$ for $i \in \{0, \ldots, 8\}$ and $b \in \{0,1\}$ such that $\mathbf{G}_{0,b} = \mathbf{BLIND}^{\mathcal{A}}_{b,\mathsf{BS}}$ and the games $\mathbf{G}_{i,b}$ and $\mathbf{G}_{i-1,b}$ are close for all $b \in \{0,1\}$. Finally, we show via a reduction from the blindness of $\mathsf{BS}[\mathsf{LF}]$ that $\mathbf{G}_{8,0}$ and $\mathbf{G}_{8,1}$ are close. Unless otherwise stated, random oracles are simulated honestly.

Game $\mathbf{G}_{0,b}$: This game is defined as the blindness game $\mathbf{BLIND}^{\mathcal{A}}_{b,\mathsf{BS}}$. We briefly recall this game and fix some notation for the rest of the proof. First, $\mathcal{A}$ outputs a public key pk and messages $\mathsf{m}_0, \mathsf{m}_1$. Then, it gets access to oracles $\mathrm{O}_0, \mathrm{O}_1$, that simulate a user with input $\mathsf{m}_b, \mathsf{m}_{1-b}$, respectively. After the interaction with these oracles, $\mathcal{A}$ gets the resulting signatures σ_0, σ_1 for the messages $\mathsf{m}_0, \mathsf{m}_1$, respectively.

Throughout the proof, we will distinguish the variables used in the oracles $\mathrm{O}_0, \mathrm{O}_1$ by superscripts $\{L, R\}$. For example, by N^L we denote the cut-and-choose parameter that $\mathcal{A}$ sends in its first message to oracle O_0. If we omit the superscript, our desciption should be understood to apply for both oracles.

By how we defined $\mathbf{G}_{0,b}$, we have

$$\epsilon = \left| \Pr\left[\mathbf{G}_{0,0} \Rightarrow 1\right] - \Pr\left[\mathbf{G}_{0,1} \Rightarrow 1\right] \right|.$$

Game $\mathbf{G}_{1,b}$: This game is identical to $\mathbf{G}_{0,b}$, except after the adversary sends the first message N in each oracle $\mathrm{O}_0, \mathrm{O}_1$, the game uniformly samples $\hat{J} \leftarrow\!\!\$\, [N]$. Later, if $\hat{J} \neq J$, the game aborts. If the game guesses J correctly for both oracles, then the game is still identical to $\mathbf{G}_{0,b}$. Furthermore, the view of $\mathcal{A}$ is independent of $\hat{J}$ before the potential abort. Hence, we have

$$\Pr[\mathbf{G}_{1,b} \Rightarrow 1] = \frac{1}{N^L N^R} \Pr[\mathbf{G}_{0,b} \Rightarrow 1].$$

In the following, we will assume that $\hat{J}$ is guessed correctly for both oracles, i.e. $\hat{J}^L = J^L$ and $\hat{J}^R = J^R$.

Game $\mathbf{G}_{2,b}$: This game is identical to $\mathbf{G}_{1,b}$, except that the we change how the game generates the values $\mathsf{prer}_j, j \in [N]$. Namely, the game computes a punctured key $k_{\hat{J}} \leftarrow \mathsf{PRF.Puncture}(k, \hat{J})$ and sets

$$\mathsf{prer}_{\hat{J}} := \mathsf{PRF.Eval}(k, \hat{J}) \text{ and } \mathsf{prer}_j := \mathsf{PRF.Eval}(k_{\hat{J}}, j) \text{ for all } j \in [N] \setminus \{\hat{J}\}.$$

Later, it returns $k_{\hat{J}}$ as its punctured key as part of the sixth message of the interaction, where we use the assumption that $\hat{J} = J$. By the completeness of PRF, this does not change the view of the adversary. Thus, we have

$$\Pr[\mathbf{G}_{2,b} \Rightarrow 1] = \Pr[\mathbf{G}_{1,b} \Rightarrow 1].$$

Game $\mathbf{G}_{3,b}$: This game is identical to $\mathbf{G}_{2,b}$ except that the value $\mathsf{prer}^L_{\hat{J}^L}$ is sampled at random as $\mathsf{prer}^L_{\hat{J}^L} \leftarrow\!\!\$\ \{0,1\}^{n_{\mathsf{PRF}}}$ instead of using $\mathsf{prer}^L_{\hat{J}^L} := \mathsf{PRF.Eval}(k^L, \hat{J}^L)$. The probability difference between $\mathbf{G}_{2,b}$ and $\mathbf{G}_{3,b}$ can be bounded by the pseudorandomness property of the puncturable pseudorandom function PRF. This can be seen with the following reduction $\mathcal{B}$.

- Get a key and messages from the adversary, i.e. $(\mathsf{pk}, \mathsf{m}_0, \mathsf{m}_1, St) \leftarrow \mathcal{A}(1^n)$.
- Run $\mathcal{A}$ on input St with access to random oracles and interactive oracles $\mathrm{O}_0, \mathrm{O}_1$, i.e. $St' \leftarrow \mathcal{A}^{\mathrm{O}_0,\mathrm{O}_1}(St)$. The oracle O_1 is provided as in game $\mathbf{G}_{2,b}$ and oracle O_0 is provided as follows:
  - When $\mathcal{A}$ sends N^L, guess $\hat{J}^L \leftarrow\!\!\$\ [N^L]$ as in game $\mathbf{G}_{2,b}$ and output $\hat{J}^L$ to the PRF challenger. Obtain the punctured key $k_{\hat{J}^L}$ and value $\mathsf{prer}^L_{\hat{J}^L}$.
  - Use $k_{\hat{J}^L}$ to sample prer^L_j for $j \in [N^L] \setminus \{\hat{J}^L\}$ as in $\mathbf{G}_{3,b}$. Continue the oracle simulation as in $\mathbf{G}_{2,b}$. According to this, if the simulation does not abort, send the key $k_{\hat{J}^L}$ in the sixth message of the interaction.
- Let σ_b, σ_{1-b} be the local outputs of $\mathrm{O}_0, \mathrm{O}_1$, respectively. If $\sigma_0 = \quad \perp$ or $\sigma_1 = \quad \perp$, then run $b' \leftarrow \mathcal{A}(St', \perp, \perp)$. Else, run $b' \leftarrow \mathcal{A}(St', \sigma_0, \sigma_1)$ and output b'.

In the case that $\mathsf{prer}^L_{\hat{J}^L}$ is uniformly sampled from $\{0,1\}^{n_{\mathsf{PRF}}}$, $\mathcal{B}$ perfectly simulates $\mathbf{G}_{3,b}$. While if $\mathsf{prer}^L_{\hat{J}^L}$ is generated from PRF, $\mathcal{B}$ perfectly simulates $\mathbf{G}_{2,b}$. Using the security of PRF with input length $\log(N^L)$, we get

$$|\Pr[\mathbf{G}_{2,b} \Rightarrow 1] - \Pr[\mathbf{G}_{3,b} \Rightarrow 1]| \leq \epsilon_{\mathsf{PRF}}.$$

Game $\mathbf{G}_{4,b}$: This game is identical to $\mathbf{G}_{3,b}$ except that $\mathsf{prer}^R_{\hat{J}^R} \leftarrow\!\!\$\ \{0,1\}^{n_{\mathsf{PRF}}}$ is sampled at random instead of generated using PRF. Similar to our previous step, we can construct a reduction against the security of PRF with input length $\log(N^R)$ with advantage at most ϵ_{PRF} and obtain

$$|\Pr[\mathbf{G}_{3,b} \Rightarrow 1] - \Pr[\mathbf{G}_{4,b} \Rightarrow 1]| \leq \epsilon_{\mathsf{PRF}}.$$

Game $\mathbf{G}_{5,b}$: This game is identical to $\mathbf{G}_{4,b}$ except that we sample the value $\mathsf{r}_{\hat{J}}$ uniformly at random instead of generating it using $\mathsf{H}_x(\mathsf{prer}_{\hat{J}})$. That is, the game samples

$$\mathsf{r}_{\hat{J}} = (\alpha_{\hat{J}}, \beta_{\hat{J}}, \varphi_{\hat{J}}, \gamma_{\hat{J}}) \leftarrow\!\!\$\ \mathcal{D} \times \mathcal{S} \times \mathcal{R}_{\mathsf{ck}} \times \{0,1\}^{n_{\mathsf{PRF}}}.$$

It is clear that $\mathcal{A}$ can only distinguish this from $\mathbf{G}_{4,b}$ if $\mathsf{H}_x(\mathsf{prer}_{\hat{J}})$ is queried. Due to the previous change, $\mathsf{prer}_{\hat{J}}$ uniformly sampled over $\{0,1\}^{n_{\mathsf{PRF}}}$ and the only information that $\mathcal{A}$ gets about $\mathsf{prer}_{\hat{J}}$ is given by its hash value. Thus, we can apply union bound over the two user oracles and all queries to H_x to obtain

$$|\Pr[\mathbf{G}_{4,b} \Rightarrow 1] - \Pr[\mathbf{G}_{5,b} \Rightarrow 1]| \leq \frac{2Q_{\mathsf{H}_x}}{2^{n_{\mathsf{PRF}}}}.$$

Game $\mathbf{G}_{6,b}$: This game is identical to $\mathbf{G}_{5,b}$ except that $\mathsf{com}_{\mathsf{r}}$ is computed differently. Concretely, the game samples $h_{\hat{J}} \leftarrow\!\!\$\ \{0,1\}^n$ and computes the $\mathsf{com}_{\mathsf{r}}$ as

$$\mathsf{com}_{\mathsf{r}} := \mathsf{H}_{\mathsf{r}}(\mathsf{H}_{\mathsf{r}}(\mathsf{r}_1), \ldots, \mathsf{H}_{\mathsf{r}}(\mathsf{r}_{\hat{J}-1}), h_{\hat{J}}, \mathsf{H}_{\mathsf{r}}(\mathsf{r}_{\hat{J}+1}), \ldots, \mathsf{H}_{\mathsf{r}}(\mathsf{r}_N))$$

Later it returns $h_{\hat{\jmath}}$ as part of its third message. It is clear that $\mathcal{A}$'s view does not change unless it queries $\mathsf{H}_\mathsf{r}(\mathsf{r}^X_{\hat{\jmath}X})$ for $X \in \{L, R\}$. Note that $\mathcal{A}$ obtains no information about $\gamma_{\hat{\jmath}}$ and $\gamma_{\hat{\jmath}}$ is sampled uniformly at random. Thus, we can apply a union bound over all $Q_{\mathsf{H}_\mathsf{r}}$ random oracle queries and $X \in \{L, R\}$ and obtain

$$|\Pr[\mathbf{G}_{6,b} \Rightarrow 1] - \Pr[\mathbf{G}_{7,b} \Rightarrow 1]| \leq \frac{2Q_{\mathsf{H}_\mathsf{r}}}{2^{n_{\mathsf{PRF}}}}.$$

Game $\mathbf{G}_{7,b}$: This game is identical to $\mathbf{G}_{6,b}$, except that we change how the commitment $\mu_{\hat{\jmath}}$ is computed. In previous games, the commitment was computed as

$$\mu_{\hat{\jmath}} := \mathsf{Translate}(\mathsf{ck}, \mu_0, \varphi_{\hat{\jmath}}) = \mathsf{Com}(\mathsf{ck}, \mathsf{m}; \varphi_0 + \varphi_{\hat{\jmath}}).$$

Instead, the game now samples $\varphi^* \leftarrow\!\!\$\, \mathcal{R}_{\mathsf{ck}}$ and computes

$$\mu_{\hat{\jmath}} := \mathsf{Com}(\mathsf{ck}, \mathsf{m}; \varphi^*).$$

When outputting the signatures, the game uses φ^* instead of $\varphi_0 + \varphi_{\hat{\jmath}}$. Here, we again use our assumption that $\hat{J} = J$. According to the aborts previously introduced, $\mathcal{A}$ has no information regarding $\varphi_{\hat{\jmath}}$. Hence, the distribution of $\varphi_0 + \varphi_{\hat{\jmath}}$ conditioned on $k_{\hat{\jmath}}, (\varphi_0 + \varphi_j)_{j \neq \hat{\jmath}}$ and φ_0 is uniformly random as $\varphi_{\hat{\jmath}}$ is uniformly random and independent of all the other values. Therefore, the view of $\mathcal{A}$ in $\mathbf{G}_{7,b}$ is the same as in $\mathbf{G}_{6,b}$, which implies that

$$\Pr[\mathbf{G}_{7,b} \Rightarrow 1] = \Pr[\mathbf{G}_{6,b} \Rightarrow 1].$$

Game $\mathbf{G}_{8,b}$: This game is identical to $\mathbf{G}_{7,b}$ except that the game now computes $\mu_0 := \mathsf{Com}(\mathsf{ck}, \bar{\mathsf{m}}, \varphi_0)$ for arbitrary (say random) message $\bar{\mathsf{m}}$. We claim that $\mathcal{A}$'s view in this game is identical to its view in $\mathbf{G}_{7,b}$. This is because in our previous changes, we established that the only information that $\mathcal{A}$ gets about φ_0 is the commitment μ_0 itself. In particular, the signatures that $\mathcal{A}$ gets are independent of φ_0^L and φ_0^R. Therefore, we can use that CMT is perfectly hiding, i.e. $\mathsf{Com}(\mathsf{ck}, \bar{\mathsf{m}}, \varphi_0)$ and $\mathsf{Com}(\mathsf{ck}, \mathsf{m}, \varphi_0)$ are identically distributed given ck. Therefore, the view of $\mathcal{A}$ does not change and we have

$$\Pr[\mathbf{G}_{8,b} \Rightarrow 1] = \Pr[\mathbf{G}_{7,b} \Rightarrow 1].$$

To summarize, so far we established the bound

$$\epsilon \leq N^L N^R \left(\frac{4(Q_{\mathsf{H}_x} + Q_{\mathsf{H}_\mathsf{r}})}{2^{n_{\mathsf{PRF}}}} + 4\epsilon_{\mathsf{PRF}} + |\Pr[\mathbf{G}_{8,0} \Rightarrow 1] - \Pr[\mathbf{G}_{8,1} \Rightarrow 1]| \right).$$

Lastly, we construct a reduction $\mathcal{B}'$ distinguishing between the two blindness games of $\mathsf{BS}[\mathsf{LF}]$ using $\mathcal{A}$ as a subroutine. This allows us to bound the advantage of $\mathcal{A}$ in distinguishing the games $\mathbf{G}_{8,0}, \mathbf{G}_{8,1}$. Reduction $\mathcal{B}'$, which runs either in game $\mathbf{BLIND}_{0,\mathsf{BS}[\mathsf{LF}]}$ or game $\mathbf{BLIND}_{1,\mathsf{BS}[\mathsf{LF}]}$, is as follows.

- $\mathcal{B}'$ runs[8] $(\mathsf{pk}, \mathsf{m}_0, \mathsf{m}_1, St) \leftarrow \mathcal{A}(\mathcal{A})$, and commits to messages $\mathsf{m}_0, \mathsf{m}_1$ via $\mu_b^* := \mathsf{Com}(\mathsf{ck}, \mathsf{m}_b; \varphi_b^*), \varphi_b^* \leftarrow\!\!\$\, \mathcal{R}_{\mathsf{ck}}$ for both $b \in \{0, 1\}$. Then, $\mathcal{B}'$ forwards the public key pk (after removing ck), and the messages μ_0^*, μ_1^* to its game.

[8] Note that this step is where we have to slightly modify $\mathcal{B}'$ for the proof of honest signer blindness or semi-honest signer blindness.

- $\mathcal{B}'$ is executed with access to oracles $\mathrm{O}'_0, \mathrm{O}'_1$. In this step, $\mathcal{B}'$ runs the adversary with access to oracles $\mathrm{O}_0, \mathrm{O}_1$ and random oracles on input St. Thereby, it provides the random oracles honestly via lazy sampling, except random oracle H, for which it forwards queries from $\mathcal{A}$ to its own game. In both oracles $\mathrm{O}_0, \mathrm{O}_1$ interacting with $\mathcal{A}$, $\mathcal{B}'$ simulates the user protocol as follows.
  - When $\mathcal{A}$ sends its first message with N, sample $\hat{J} \leftarrow_\$ [N]$. Generate keys $k \leftarrow \mathsf{PRF.Gen}(1^{n_{\mathsf{PRF}}}, 1^{\log N})$ and $k_{\hat{J}} \leftarrow \mathsf{PRF.Puncture}(k, \hat{J})$. Set $\mathsf{prer}_j := \mathsf{PRF.Eval}(k_{\hat{J}}, j)$ and set $\mathsf{r}_j := \mathsf{H}_x(\mathsf{prer}_j)$ for all $j \in [N] \setminus \{\hat{J}\}$. Then, set the commitment $\mu_0 := \mathsf{Com}(\mathsf{ck}, \bar{\mathsf{m}}, \varphi_0)$ for an arbitrary message $\bar{m}$ and $\varphi_0 \leftarrow_\$ \mathcal{R}_{\mathsf{ck}}$. Sample the value $h_{\hat{J}} \leftarrow_\$ \{0,1\}^n$. Other values are calculated based on the original protocol with these changes. Then, $\mu_0, \mathsf{com}_{\mathsf{r}}$ to $\mathcal{A}$.
  - Upon receiving, $R_1, \ldots, R_l$ where $l = \log N$, calculate $\tilde{R}_{\hat{J}}$ as in the protocol and forward that to the corresponding oracle (O'_b for O_b). Then set $c_{\hat{J}}$ as the response from this oracle. For other sessions $j \in [N] \setminus \{\hat{J}\}$, run the user protocol honestly.
  - Follow the protocol until receiving J. If $J \neq \hat{J}$, abort the entire execution. Else, follow the protocol honestly until receiving $s_{\hat{J}}$ from $\mathcal{A}$. Then, forward this value to the corresponding oracle (O'_b for O_b).
- $\mathcal{B}'$ receives signatures $(c'_0, s'_0), (c'_1, s'_1)$ from its own game. It runs the adversary $\mathcal{A}$ on input $\sigma_0 := (s'_0, c'_0, \varphi^*_0)$ and $\sigma_1 := (s'_1, c'_1, \varphi^*_1)$ and gets a bit $b' \in \{0,1\}$ in return. $\mathcal{B}$ outputs this bit b'.

We can easily see that if $\mathcal{B}'$ runs in $\mathbf{BLIND}_{0,\mathsf{BS[LF]}}$, $\mathcal{B}'$ perfectly simulates $\mathbf{G}_{8,0}$ for $\mathcal{A}$. Vice versa, if the underlying game is $\mathbf{BLIND}_{1,\mathsf{BS[LF]}}$, $\mathcal{B}'$ perfectly simulates $\mathbf{G}_{8,1}$. Hence, denoting the advantage of $\mathcal{B}'$ by $\epsilon_{\mathsf{BS[LF]}}$, we get

$$|\Pr[\mathbf{G}_{8,0} \Rightarrow 1] - \Pr[\mathbf{G}_{8,1} \Rightarrow 1]| \leq \epsilon_{\mathsf{BS[LF]}}.$$

□

G.2 One-More Unforgeability

Proof (of Theorem 2). Let $\mathcal{A}$ be an adversary in the one-more unforgeability game of $\mathsf{CCCBS[LF]}$. Suppose that $\mathcal{A}$ runs in in time t, initiates the protocol at most p times, makes at most $Q_{\mathsf{H}}, Q_{\mathsf{H_r}}, Q_{\mathsf{H}_c}$ queries to $\mathsf{H}, \mathsf{H_r}, \mathsf{H}_c$ respectively, and has success probability ϵ.

Game $\mathbf{G}_0$: We start with $\mathbf{G}_0$, which is the ℓ-one-more unforgeability game, i.e. $\mathbf{G}_0 = \ell\text{-}\mathbf{OMUF}^{\mathcal{A}}_{\mathsf{CCCBS[LF]}}$. The success probability of $\mathcal{A}$ in this game is ϵ. Recall that in this game, keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^n)$ are sampled and pk is given to $\mathcal{A}$. Then, $\mathcal{A}$ gets access to an oracle O that simulates the signer algorithm $\mathsf{S}(\mathsf{sk})$. In the end, $\mathcal{A}$ outputs messages and signatures (m_i, σ_i) and the game outputs 1 if these are valid signatures on distinct messages, and there are more pairs in the output than the number of completed interactions with oracle O.

For $t \in \{\mathsf{r}, c\}$, we say that a value y is *extractable* (at some point of the execution) *from a random oracle* H_t if there is a query v to H_t such that $\mathsf{H}_t(v) = y$ (and the query happened before that point). In this case, note that whenever $\mathcal{A}$

sends a value y that is extractable, the game can find such a query v by searching in the list of random oracle queries. We also say that y is *extractable to* v.

Game $\mathbf{G}_1$: This game is identical to $\mathbf{G}_0$, except the game aborts when at least one of the following bad events occurs:

(1) There are collisions in $\mathsf{H}_\mathsf{r}, \mathsf{H}_c$ i.e. there exists two queries $x \neq x'$ to H_t such that $\mathsf{H}_t(x) = \mathsf{H}_t(x')$ for $t \in \{\mathsf{r}, c\}$. Clearly, the probability of this happening is at most $(Q^2_{\mathsf{H}_\mathsf{r}} + Q^2_{\mathsf{H}_c})/2^n$.

(2) When com_r is sent by $\mathcal{A}$ in an interaction, com_r is not extractable from H_r or it is extractable to h_j's, but there exists some j such that h_j is not extractable from H_r, and later even with $j \neq J$, the signer does not abort (i.e. algorithm Check outputs 1). In other words, this event occurs if the game can not find values $\bar{\mathsf{r}}_i$ in the list of random oracle queries such that

$$\bar{\mathsf{com}}_\mathsf{r} = \mathsf{H}_\mathsf{r}(\mathsf{H}_\mathsf{r}(\bar{\mathsf{r}}_1), \ldots, \mathsf{H}_\mathsf{r}(\bar{\mathsf{r}}_N)).$$

Clearly, once com_r is fixed, this event can only happen when the adversary finds the correct $(h_1, \ldots, h_N)$ or r_j such that H_r outputs com_r or h_j, respectively. Since the adversary makes at most $Q_{\mathsf{H}_\mathsf{r}}$ queries to H_r, the probability of this happening is $Q_{\mathsf{H}_\mathsf{r}}/2^n$ in a protocol execution. Hence, by union bound, the probability of this event is $pQ_{\mathsf{H}_\mathsf{r}}/2^n$.

(3) When com_c is sent by $\mathcal{A}$ in an interaction, com_c is not extractable from H_c when received, but the signer does not abort later. Note that algorithm Check verifies that $\mathsf{com}_c = \mathsf{H}_c(c_1, \ldots, c_N)$. Thus, this event can only occur if $\mathcal{A}$ finds preimages of com_c after com_c is fixed. Using a union bound over the p interactions and Q_{H_c} hash queries we can bound the probability of this event by $pQ_{\mathsf{H}_c}/2^n$.

(4) The final output of $\mathcal{A}$ contains two pairs $(\mathsf{m}_0, \sigma_0 = (c'_0, s'_0, \varphi_0)), (\mathsf{m}_1, \sigma_1 = (c'_1, s'_1, \varphi_1))$ such that $\mathsf{m}_0 \neq \mathsf{m}_1$ but $\mathsf{Com}(\mathsf{ck}, \mathsf{m}_0; \varphi_0) = \mathsf{Com}(\mathsf{ck}, \mathsf{m}_1; \varphi_1)$. We can construct a straight-forward reduction $\mathcal{B}_{\mathsf{CMT}}$ against the binding property of CMT with ϵ_{CMT} advantage to bound the probability of this event.

In summary, we obtain

$$|\Pr[\mathbf{G}_0 \Rightarrow 1] - \Pr[\mathbf{G}_1 \Rightarrow 1]| \leq \frac{Q^2_{\mathsf{H}_\mathsf{r}} + Q^2_{\mathsf{H}_c} + p \cdot Q_{\mathsf{H}_\mathsf{r}} + p \cdot Q_{\mathsf{H}_c}}{2^n} + \epsilon_{\mathsf{CMT}}.$$

The purpose of the introduced aborts is to (1) ensure that each hash value corresponds to a unique query, (2 and 3) rule out that the adversary cheats by not sending correct hashes and still providing valid preimages later, and (4) prohibit the adversary in outputting two distinct message-signature pairs for which the commitments of the messages are identical.

We emphasize that due to (1), we can assume that if the game can extract com_c or com_r, the values that are sent later by $\mathcal{A}$ (i.e. values $c_1, \ldots, c_N$ and $\mathsf{r}_1, \ldots, \mathsf{r}_N$) are the ones that the game could extract before.

Now, we define the notion of successful cheating. Namely, we say that the adversary $\mathcal{A}$ *successfully cheats* in a signing interaction, if neither the signer nor the game abort, but one of the following occurs:

(1) com_r, when received, is extractable from H_r into $h_1, \ldots, h_N$, but for some $j \in [N]$, h_j is not extractable from H_r into r_j, or
(2) Although com_r is extractable from H_r into $\mathsf{r}_j = (\alpha_j, \beta_j, \varphi_j, \gamma_j)$ and com_c is extractable from H_c into c_j's, we have

$$c_j \neq \mathsf{H}\left(\mu_j, \tilde{R}_j + \mathsf{F}(\alpha_j) + \beta_j \cdot \mathsf{pk}'\right) + \beta_j$$

for $\mu_j = \mathsf{Translate}(\mathsf{ck}, \mu_0, \varphi_j)$.

In $\mathbf{G}_1$, $\mathcal{A}$ can successfully cheat only if it guesses J correctly. Hence, in a signing interaction, the probability that $\mathcal{A}$ successfully cheats is at most $1/N$.

Next, we will upper-bound the expected number of times $\mathcal{A}$ successfully cheats in $\mathbf{G}_1$. 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 $\mathbf{G}_1$ is at most

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

Game $\mathbf{G}_2$: This game is identical to $\mathbf{G}_1$, except that the game aborts when the adversary successfully cheats more than $\lambda = 3\lceil \log p \rceil + \log(2/\epsilon)$ times. We can upper-bound the probability that $\mathcal{A}$ cheats more than λ times in $\mathbf{G}_1$ by $\epsilon/2$ as follows.

Let X be a random variable for the number of times $\mathcal{A}$ successfully cheats in $\mathbf{G}_1$ and k be the value such that at protocol execution p, $N = 2^k - 2$. By the analysis above, $k \leq \lceil \log p \rceil + 1$. Denoting by Bernoulli($\phi$) a random variable with Bernoulli distribution and parameter ϕ, we can bound the random variable X by a sum of independent identically distributed Bernoulli random variables. Precisely, we have

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

Let Y be this sum of Bernoulli random variables. We can now use Lemma 5 with $\mathbb{E}[Y] = \lceil \log p \rceil$ and $s = \lambda$ and get

$$\Pr[X \geq \lambda] \leq \Pr[Y \geq \lambda] \leq \exp\left(3\,\mathbb{E}[Y] - \lambda\right) = \frac{\epsilon}{2}.$$

Thus, the probability of the abort we introduced is at most $\epsilon/2$, which implies that

$$\left|\Pr\left[\mathbf{G}_1 \Rightarrow 1\right] - \Pr\left[\mathbf{G}_2 \Rightarrow 1\right]\right| \leq \frac{\epsilon}{2}.$$

Game $\mathbf{G}_3$: This game is identical to $\mathbf{G}_2$ except that, after the signer oracle receives com_r in each protocol execution, the game does the following:

1. Sample a random session $\hat{j} \leftarrow\!\!\$\, [N]$.
2. If com_r or $h_{\hat{j}}$ is not extractable from H_r, set $C_{\hat{j}} := \perp$. Otherwise, if com_r is extractable from H_r to $(h_1, \ldots, h_N)$ and $h_{\hat{j}}$ is extractable from H_r to $\mathsf{r}_{\hat{j}}$, do the following:
   (a) Sample a random scalar $C_{\hat{j}} \leftarrow\!\!\$\, \mathcal{S}$.
   (b) Pick an index $i' \in S_{\hat{j}} = \{i \in [l] : i^{th}\text{-bit of } \hat{j} \text{ is } 1\}$.
   (c) Sample $r_{i'} \leftarrow\!\!\$\, \mathcal{D}$ and set $R_{i'} := \mathsf{F}(r_{i'}) + C_{\hat{j}} \cdot (-\mathsf{pk}')$.
   (d) After initializing R_i for $i \in [l] \setminus \{i'\}$ as in the signing protocol, calculate $\tilde{R}'_{\hat{j}}$ (as the user does in the protocol) using the extracted randomness $\mathsf{r}_{\hat{j}}$. Then, if the hash value $\mathsf{H}(\mu_{\hat{j}}, \tilde{R}'_{\hat{j}})$ is already defined, the game aborts. If not, set $\mathsf{H}(\mu_{\hat{j}}, \tilde{R}'_{\hat{j}}) := C_{\hat{j}} - \beta_{\hat{j}}$.
   (e) Later when s_J needs to be calculated and $i' \in S_J = \{i \in [l] : i^{th}\text{-bit of } J \text{ is } 1\}$, use $r_{i'} + C_{\hat{j}} \cdot (-\mathsf{sk})$ as the preimage of $R_{i'}$.

   Terminology: We call the above session $\hat{j}$ that has H defined in this way as a *programmed session.*
3. For each $j \in [N] \setminus \{\hat{j}\}$, if com_r is not extractable from H_r, or it is extractable from H_r to $(h_1, \ldots, h_N)$, but h_j is not extractable from H_r, set $C_j := \perp$. Otherwise, h_j is extractable into r_j. After initializing R_i for each $i \in [l]$ as in the signing protocol except for $R_{i'}$ if i' is defined in the case above, compute $\tilde{R}'_j$ (as the user does in the protocol) and check if $\mathsf{H}(\mu_j, \tilde{R}'_j)$ is already defined. If it is, the game aborts. Else, set $\mathsf{H}(\mu_j, \tilde{R}'_j)$ uniformly at random and let $C_j := \mathsf{H}(\mu_j, \tilde{R}'_j) + \beta_j$.
4. Send $R_1, \ldots, R_l$ to $\mathcal{A}$ after all of the above is done.

Assuming none of the aborts that we introduced occurs, we can show that the view of $\mathcal{A}$ does not change. To see this, first, the outputs of the random oracle H are still uniformly distributed in $\mathcal{S}$. Moreover, the distribution of $R_1, \ldots, R_l$ stays the same because the distribution of $R_{i'} = \mathsf{F}(r_{i'}) + C_{\hat{j}} \cdot (-\mathsf{pk}') = \mathsf{F}(r_{i'} + C_{\hat{j}} \cdot (-\mathsf{sk}'))$ is the same as $\mathsf{F}(r_{i'})$ given that $r_{i'}$ is uniformly generated. Furthermore, for the programmed session $\hat{j}$, the joint distribution of $(\tilde{R}'_{\hat{j}}, c_{\hat{j}}, s_{\hat{j}})$ remains the same because $\tilde{R}'_{\hat{j}} = \mathsf{F}(s_{\hat{j}}) + c_{\hat{j}} \cdot \mathsf{pk}'$ in both games.

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

(1) (μ, R) has been queried by $\mathcal{A}$. The game tries to program the oracle with $R = \sum_{i \in S_j} R_i + \mathsf{F}(\alpha_j) + \beta_j \cdot \mathsf{pk}'$ for some $j \in [N]$, and $R_i = \mathsf{F}(r_i)$ is uniform in $\mathcal{R}$ as r_i is uniform in $\mathcal{D}$ and the linear function is smooth. Also, the value of R that $\mathcal{A}$ queried the random oracle is independent of R_i because R_i was not revealed to the adversary yet. Thus, this happens with probability at most $1/|\mathcal{R}|$.

(2) (μ, R) has been programmed in another protocol execution. The same argument from above applies as R_i is independent from other values in a different protocol execution. Hence, this happens with probability at most $1/|\mathcal{R}|$.
(3) (μ, R) has been programmed in the same protocol execution. Let the two sessions be $j_1, j_2 \in [N]$ and $j_1 \neq j_2$. Then,

$$R = \sum_{i \in S_{j_1}} R_i + \mathsf{F}(\alpha_{j_1}) + \beta_{j_1} \cdot \mathsf{pk}' = \sum_{i \in S_{j_2}} R_i + \mathsf{F}(\alpha_{j_2}) + \beta_{j_2} \cdot \mathsf{pk}'.$$

Since $j_1 \neq j_2$, we know that $S_{j_1} \neq S_{j_2}$. Thus, there is some i which is in only one of S_{j_1}, S_{j_2}. Thus, R_i is independent from any other values involved in the event. Then the above probability can happen with probability at most $1/|\mathcal{R}|$.

Thus, $\mathcal{A}$'s success probability is reduced by at most $p^2(p^2 + Q_{\mathsf{H}})/|\mathcal{R}|$ where p^2 upper-bounds the number of times the game tries to program H (in each signing protocol execution, the game tries to programs H $N \leq p$ times, so over all executions the game tries to program H at most p^2 times) and $p^2 + Q_{\mathsf{H}}$ upper-bounds the number of times H is programmed or queried. Therefore, we have

$$|\Pr[\mathbf{G}_2 \Rightarrow 1] - \Pr[\mathbf{G}_3 \Rightarrow 1]| \leq \frac{p^2(p^2 + Q_{\mathsf{H}})}{|\mathcal{R}|}.$$

Game $\mathbf{G}_4$: This game is identical to $\mathbf{G}_3$, except how the signer generates random cut-and-choose index J. Concretely, after receiving com_c, if com_c is extractable from H_c into $c_1, \ldots, c_N$ and for all $j \in [N]$, $c_j = C_j$ i.e. there is no cheating from the adversary, set $J := \hat{j}$ where $\hat{j}$ is the programmed session defined in $\mathbf{G}_3$. Otherwise, if com_c is not extractable or extractable from H_c into $c_1, \ldots, c_N$, but $c_j \neq C_j$ for some $j \in [N]$, set $J := (N+1) \oplus \hat{j} = (2^l - 1) \oplus \hat{j}$ which is the bit-wise complement of $\hat{j}$.

Since $\hat{j}$ is uniformly randomly generated from $[N]$, J's distribution remains uniform in both cases as $N = 2^l - 2$. Hence, the success probability of $\mathcal{A}$ remains the same as $\mathbf{G}_3$, i.e.

$$\Pr[\mathbf{G}_4 \Rightarrow 1] = \Pr[\mathbf{G}_3 \Rightarrow 1].$$

Next, we define new nomenclature regarding the programmed sessions and the signatures related to them. We call a programmed session $\hat{j}$, a *completed programmed session*, if $J = \hat{j}$ and the signer does not abort the signing protocol. Also, we call a valid signature $\sigma = (c', s', \varphi')$ for a message m, a *fake signature*, if with $R' := \mathsf{F}(s') - c' \cdot \mathsf{pk}'$ and $\mu := \mathsf{Com}(\mathsf{ck}, \mathsf{m}; \varphi')$, there is a programmed session (not necessarily a completed one) that programs H at (μ, R').
Game $\mathbf{G}_5$: This game is identical to $\mathbf{G}_4$ except that, after the adversary outputs the signatures, the game aborts if there are more fake signatures than the number of completed programmed sessions. We can bound the probability that this bad event occurs, using the following claim:

Claim. There is a reduction $\mathcal{B}'$ against the preimage-resistance of LF with running time t and advantage ϵ_{LF} such that $\Pr[E] \leq p \cdot \epsilon_{\mathsf{LF}}$, where E denotes the event that

$\mathbf{G}_4$ outputs 1 and adversary $\mathcal{A}$ outputs more fake signatures than the number of completed programmed sessions.

To prove the claim, we first observe that by the changes we introduced in $\mathbf{G}_1$, $\mathcal{A}$ cannot output two pairs $(\mathsf{m}_0, \sigma_0 = (c'_0, s'_0, \varphi_0)), (\mathsf{m}_1, \sigma_1 = (c'_1, s'_1, \varphi_1))$ such that $\mathsf{m}_0 \neq \mathsf{m}_1$ but $\mathsf{Com}(\mathsf{ck}, \mathsf{m}_0; \varphi_0) = \mathsf{Com}(\mathsf{ck}, \mathsf{m}_1; \varphi_1)$. This means that any message-signature pair that $\mathcal{A}$ outputs will correspond to distinct random oracle query. Hence, if $\mathcal{A}$ outputs more fake signatures than completed programmed sessions, then there will be fake signatures that does not correspond to any completed programmed sessions. In other words, there will be a fake signature that corresponds to a programmed session that is not completed.

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

1. $\mathcal{B}'$ receives the input (par, R) from the game of preimage resistant game for LF. $\mathcal{B}'$ then picks a random $k \leftarrow\!\!\$\, [p]$. $\mathcal{B}'$ simulates all interactions of $\mathcal{A}$ with the signer oracle O as in $\mathbf{G}_4$, except execution k. In the k^{th} execution, instead of setting (with the notation of $\mathbf{G}_3$) $R_{i'} := \mathsf{F}(r_{i'}) + C_{\hat{j}} \cdot (-\mathsf{pk}')$, it sets $R_{i'} := R + C_{\hat{j}} \cdot (-\mathsf{pk}')$. Recall that i'^{th} bit of $\hat{j}$ is 1, so
$$\tilde{R}_{\hat{j}} = \sum_{i \in S_{\hat{j}}} R_i = R + C_{\hat{j}} \cdot (-\mathsf{pk}') + \sum_{i \in S_{\hat{j}} \setminus \{i'\}} R_i.$$
If $\mathcal{B}'$ needs to calculate $s_{\hat{j}}$, i.e. it has to complete the programmed session, it aborts.
2. Later, if $\mathcal{A}$ succeeds and outputs more fake signatures than the number of completed programmed sessions, and the first fake signature σ for a message m 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 the signature be $\sigma = (c', s', \varphi)$. Further, define $R' := \mathsf{F}(s') - c' \cdot \mathsf{pk}'$, and $r' := \sum_{i \in S_{\hat{j}} \setminus \{i'\}} r_i$. Then, we have
$$\begin{aligned} R' = \tilde{R}'_{\hat{j}} &= \mathsf{F}(\alpha_{\hat{j}}) + \beta_{\hat{j}} \cdot \mathsf{pk}' + \tilde{R}_{\hat{j}} \\ &= \mathsf{F}(\alpha_{\hat{j}}) + \beta_{\hat{j}} \cdot \mathsf{pk}' + R + C_{\hat{j}} \cdot (-\mathsf{pk}') + \textstyle\sum_{i \in S_{\hat{j}} \setminus \{i'\}} R_i \\ &= \mathsf{F}(\alpha_{\hat{j}}) + \beta_{\hat{j}} \cdot \mathsf{pk}' + R + C_{\hat{j}} \cdot (-\mathsf{pk}') + \mathsf{F}(r'). \end{aligned}$$
Using $\mathsf{F}(s') - c' \cdot \mathsf{pk}'$ and rearranging terms, we get
$$\begin{aligned} R &= \mathsf{F}(s') - c' \cdot \mathsf{pk}' - \mathsf{F}(\alpha_{\hat{j}}) - \beta_{\hat{j}} \cdot \mathsf{pk}' + C_{\hat{j}} \cdot \mathsf{pk}' - \mathsf{F}(r') \\ &= \mathsf{F}(s' - \alpha_{\hat{j}} - r') - \beta_{\hat{j}} \cdot \mathsf{pk}' + C_{\hat{j}} \cdot \mathsf{pk}' - c' \cdot \mathsf{pk}' \\ &= \mathsf{F}(s' - \alpha_{\hat{j}} - r' - \Psi(\mathsf{pk}', C_{\hat{j}}, -c')), \end{aligned}$$
where we used $\beta_{\hat{j}} = C_{\hat{j}} - c'$. Note that in the equation above, $\mathcal{B}'$ knows the values $s', \alpha_{\hat{j}}, r', \mathsf{pk}', C_{\hat{j}}, -c'$. Therefore, $\mathcal{B}'$ can return the value
$$s' - \alpha_{\hat{j}} - r' - \Psi(\mathsf{pk}', C_{\hat{j}}, -c')$$
as a preimage of R.

Therefore, $\mathcal{B}'$ can find the preimage of R if it guesses the execution that corresponds to the first fake signature in the output of $\mathcal{A}$. Note that until an abort happens, the execution k is hidden from $\mathcal{A}$. Thus, $\epsilon_{\mathsf{LF}} \geq \Pr[E]/p$ and we get

$$|\Pr[\mathbf{G}_4 \Rightarrow 1] - \Pr[\mathbf{G}_5 \Rightarrow 1]| \leq \Pr[E] \leq p \cdot \epsilon_{\mathsf{LF}}.$$

Lastly, we will construct a reduction $\mathcal{B}$ against the λ-one-more unforgeability of $\mathsf{BS}[\mathsf{LF}]$ where $\lambda = 3\lceil \log p \rceil + \log(2/\epsilon)$, simulating $\mathbf{G}_5$, and using adversary $\mathcal{A}$ as a subroutine. Reduction $\mathcal{B}$ is defined as follows:

1. $\mathcal{B}$ simulates all random oracles exept H honestly via lazy sampling. For queries to H, $\mathcal{B}$ forwards the query to its challenger and sends back the output 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 H in $\mathbf{G}_3$.
2. To start the game, $\mathcal{B}$ receives the public key pk', generates other parameters used in $\mathsf{CCCBS}[\mathsf{LF}]$ and forwards them to $\mathcal{A}$.
3. To simulate the signer oracle, $\mathcal{B}$ works as follows. For each signing protocol execution that is initiated by $\mathcal{A}$, $\mathcal{B}$ replicates $\mathbf{G}_5$ and additionally does the following:
   - After $\mathcal{A}$ initiates a protocol execution, $\mathcal{B}$ starts a signing interaction for scheme $\mathsf{BS}[\mathsf{LF}]$ with its own game and receives R^*. After initializing $\hat{j} \leftarrow\!\!\$\, [N]$ and $R_{i'}$ according to $\mathbf{G}_5$, it picks another i'' such that i''^{th} bit of $\hat{j}$ is 0 and sets $R_{i''} := R^*$.
     Then, $\mathcal{B}$ interacts with $\mathcal{A}$ by following the specification of $\mathbf{G}_5$.
   - When the protocol requires $\mathcal{B}$ to send s_J to $\mathcal{A}$, $\mathcal{B}$'s behavior will depend on the value of J.
     If $J = \hat{j}$, $\mathcal{B}$ sends $s_J := \sum_{i \in S_J} r_i = \sum_{i \in S_J} r_i + C_J \cdot (-\mathsf{sk}') + c_J \cdot \mathsf{sk}'$. This doesn't require $\mathcal{B}$ to know the secret key sk' because $c_J = C_J$ by how H is programmed in $\mathbf{G}_3$ and how J is selected in $\mathbf{G}_4$.
     If $J = (N+1) \oplus \hat{j}$, $\mathcal{B}$ forwards c_J to its own game. Upon receiving s^*, $\mathcal{B}$ sends $s_J := s^* + \sum_{i \in S_J \setminus \{i''\}} r_i$ to $\mathcal{A}$.
4. After $\mathcal{A}$ sends $\ell + 1$ valid pairs of the form $(\mathsf{m}_i, \sigma_i = (c'_i, s'_i, \varphi_i))$ to $\mathcal{B}$, $\mathcal{B}$ identifies and removes all the fake signatures. Then, $\mathcal{B}$ outputs the remaining signatures on the induced commitments, i.e. it outputs the $(\mu_i, \tilde{\sigma}_i)$ where

$$\mu_i := \mathsf{Com}(\mathsf{ck}, \mathsf{m}_i; \varphi_i), \quad \tilde{\sigma}_i := (c'_i, s'_i).$$

We see that $\mathcal{B}$ perfectly simulates $\mathbf{G}_5$ for $\mathcal{A}$. Further, using the analysis above (see $\mathbf{G}_1$), we know that the $\ell + 1$ pairs of the form $(\mathsf{m}, \sigma = (c', s', \varphi))$ that $\mathcal{A}$ outputs contain no two pairs leading to the same commitment $\mu = \mathsf{Com}(\mathsf{ck}, \mathsf{m}; \varphi)$. Therefore, all committed messages μ_i that $\mathcal{B}$ outputs are distinct. Also, the number of fake signatures is at most the number of completed programmed sessions which is at most $\ell - \mathsf{cheat}$, where 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 executions that $\mathcal{B}$ completes with its own game is equal to the number of times $\mathcal{A}$ successfully cheats, $\mathcal{B}$ outputs enough signatures.

Since $\mathcal{B}$ is simulating $\mathbf{G}_5$, the the number of times $\mathcal{A}$ successfully cheats is upper-bounded by $\lambda = \lceil \log p \rceil + \log(2/\epsilon)$. Therefore, $\mathcal{B}$ successfully breaks λ-one-more unforgeability of $\mathsf{BS}[\mathsf{LF}]$ with the same probability as $\mathcal{A}$'s success probability in $\mathbf{G}_5$, proving the theorem. □

H A Concrete Scheme based on RSA

We construct a blind signature scheme $\mathsf{BS}_{\mathsf{RSA}}$ based on the RSA assumption. Our scheme is based on the Okamoto-Guillou-Quisquater (OGQ) [31] linear function. The function is specified by public parameters $\mathsf{par} = (N, a, \lambda)$ where p and q are distinct n-bit primes and $N = pq$, $a \leftarrow\!\!\$\, \mathbb{Z}_N^*$ is sampled uniformly at random, and λ is a prime with $\gcd(N, \lambda) = \gcd(\varphi(N), \lambda) = 1$. Further, define a trapdoor $\mathsf{td} := (p, q)$. Throughout this section, we assume that these parameters are output by some setup algorithm RSAGen.

Let $\mathcal{D} := \mathbb{Z}_\lambda \times \mathbb{Z}_N^*$. It can be shown [25] that $\mathcal{D}$ forms a group with respect to the group operation

$$(x_1, y_1) \circ (x_2, y_2) := \left(x_1 + x_2 \bmod \lambda, y_1 \cdot y_2 \cdot a^{\lfloor \frac{x_1 + x_2}{\lambda} \rfloor} \bmod N \right).$$

We specify a linear function F as follows:

$$\mathsf{F} : \mathcal{D} \to \mathbb{Z}_N^*, \quad (x, y) \mapsto a^x y^\lambda \bmod N.$$

In addition, we specify a function

$$\Psi : \mathbb{Z}_N^* \times \mathbb{Z}_\lambda \times \mathbb{Z}_\lambda \to \mathcal{D}, \quad (x, s, s') \mapsto (0, x^{\lfloor -\frac{s+s'}{\lambda} \rfloor} \bmod N).$$

These funcions satisfy

$$\forall x, y \in \mathcal{D}, s \in \mathbb{Z}_\lambda : \mathsf{F}(x^s \circ y) = \mathsf{F}(x)^s \cdot \mathsf{F}(y),$$
$$\forall y \in \mathbb{Z}_N^*, s, s' \in \mathbb{Z}_\lambda : y^{s+s'} = y^s \cdot y^{s'} \cdot \mathsf{F}(\Psi(y, s, s')).$$

The collision resistance and one-wayness of the function F is tightly implied by the RSA assumption. For more details, see [25].

H.1 Invertibility of the OGQ Linear Function

We argue that the trapdoor can be used to sample uniform preimages for F. To this end, we specify an algorithm $\mathsf{Invert}(\mathsf{td}, z)$ for $z \in \mathbb{Z}_N^*$, which works as follows:

- Use p and q to compute $\rho \in \mathbb{Z}$ such that $\rho\lambda \bmod \varphi(N) = 1$.
- Sample $x \leftarrow\!\!\$\, \mathbb{Z}_\lambda$ and set $y := (za^{-x})^\rho \bmod N$. Return (x, y).

In the following, we argue that Invert outputs properly distributed preimages.

It is clear that for $(x, y) \leftarrow \mathsf{Invert}(\mathsf{td}, z)$ we have

$$\mathsf{F}(x, y) = a^x y^\lambda = a^x (za^{-x})^{\lambda\rho \bmod \varphi(N)} = z \pmod N.$$

Thus, it remains to show that the distributions

$$\mathcal{D}_1 := \big\{((x,y),z)\big|(x,y) \leftarrow\!\!\$\, \mathbb{Z}_\lambda \times \mathbb{Z}_N^*, z := a^x y^\lambda \bmod N\big\}$$

and

$$\mathcal{D}_2 := \big\{((x,y),z)\big|z \leftarrow\!\!\$\, \mathbb{Z}_N^*, x \leftarrow\!\!\$\, \mathbb{Z}_\lambda, y := (za^{-x})^\rho \bmod N\big\}$$

are the same. Fix $(x_0, y_0, z_0) \in \mathbb{Z}_\lambda \times \mathbb{Z}_N^* \times \mathbb{Z}_N^*$. As a is invertible and $y \mapsto y^\lambda$ defines a permutation on $\mathbb{Z}_N^*$, we have

$$\Pr_{(x,y,z)\leftarrow\mathcal{D}_1}[z = z_0] = \frac{1}{\varphi(N)} = \Pr_{(x,y,z)\leftarrow\mathcal{D}_2}[z = z_0].$$

By conditioning on $z = z_0$ we see that it remains to show that

$$\Pr_{(x,y,z)\leftarrow\mathcal{D}_1}[(x,y) = (x_0,y_0) \mid z = z_0] = \Pr_{(x,y,z)\leftarrow\mathcal{D}_2}[(x,y) = (x_0,y_0) \mid z = z_0].$$

Here, the left-hand side is equal to $1/\lambda$ if $z_0 = a^{x_0} y_0^\lambda \bmod N$ and 0 otherwise. The right-hand side is equal to $1/\lambda$ if $y_0 = (z_0 a^{-x_0})^\rho \bmod N$ and 0 otherwise. As both conditions are equivalent, we can conclude the analysis.

H.2 The Boosting Transform

We revisit the boosting transform introduced in [30] for the special case of the OGQ linear function. The boosting transform defines a blind signature scheme CCBS as follows.

Key Generation. Algorithm $\mathsf{CCBS.Gen}(1^n)$ generates keys as:

1. Generate parameters $\mathsf{par} = (N, a, \lambda)$ as above.
2. Sample $\mathsf{sk}' \leftarrow\!\!\$\, \mathcal{D}$, set $\mathsf{pk}' := \mathsf{F}(\mathsf{sk}')$.
3. Return the public key $\mathsf{pk} := (\mathsf{par}, \mathsf{pk}')$ and the secret key $\mathsf{sk} := \mathsf{sk}'$.

Signature Issuing. The signature issuing protocol of the scheme is presented in Figure 6. Here, the signer is stateful and its state ctr is intitialized as $\mathsf{ctr} := 1$.

Verification. A signature $\sigma = (c', s', \varphi^*)$ is verified with respect to a message m via algorithm $\mathsf{CCBS.Ver}(\mathsf{pk} = (\mathsf{par}, \mathsf{pk}'), \mathsf{m}, \sigma)$, which is as follows:

1. Compute the commitment $\mu^* := \hat{\mathsf{H}}(\mathsf{m}, \varphi^*)$
2. Return 1 if $c' = \mathsf{H}(\mu^*, \mathsf{F}(s') \cdot \mathsf{pk}'^{-c'})$. Otherwise return 0.

We highlight that for the proof of one-more unforgeability in [30] it is not important that the commitments μ_i are computed using a random oracle. In fact, what it needed is only that this commitment is binding. Indeed, it is easy to see that the proof goes through using any binding commitment scheme. We denote this modified scheme using a commitment scheme CMT by $\mathsf{CCBS}[\mathsf{CMT}]$. We summarize the one-more unforgeability bounds of the scheme $\mathsf{CCBS}[\mathsf{CMT}]$ in the following theorem. The concrete bounds can easily be derived from [25,30].

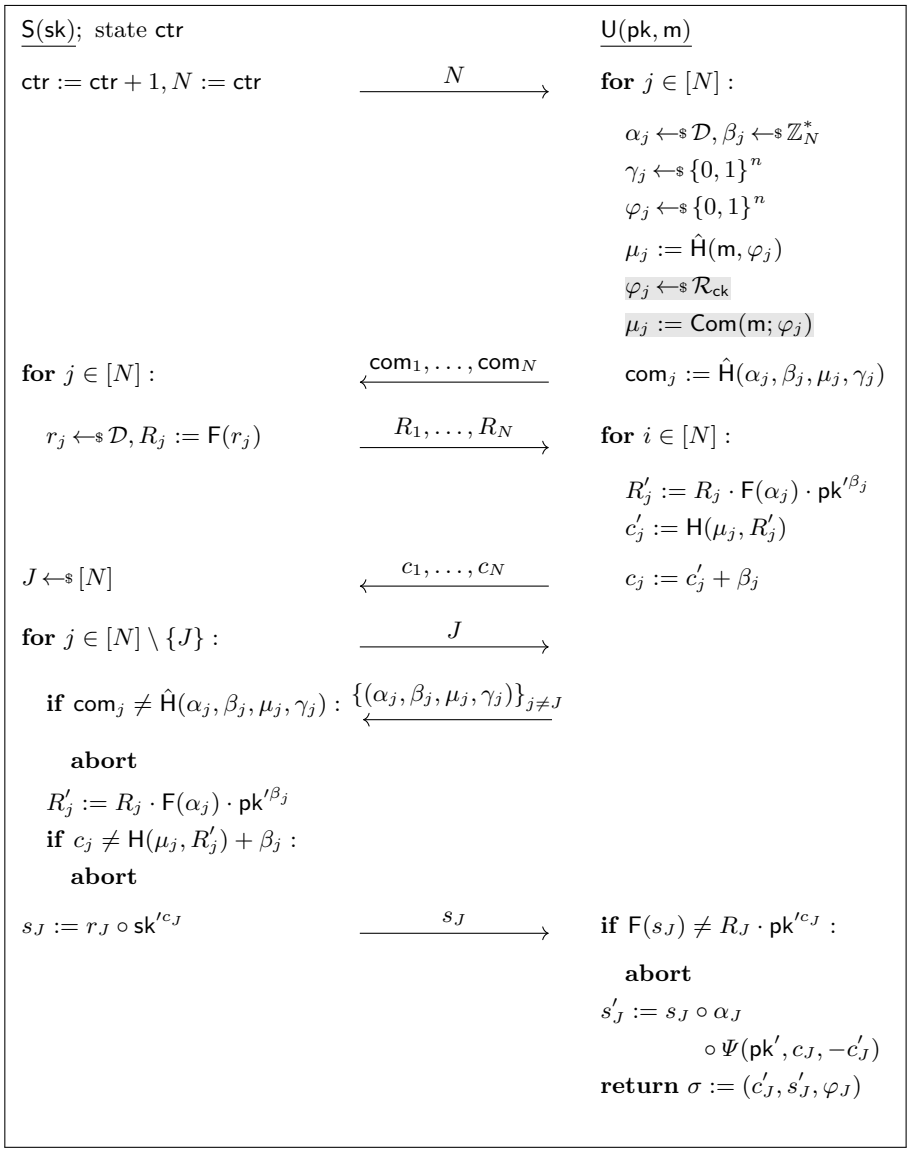

Fig. 6. The signature issuing protocol of the blind signature scheme CCBS obtained via the boosting construction, where $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_\lambda, \hat{\mathsf{H}} : \{0,1\}^* \to \{0,1\}^n$ are random oracles. The state ctr of S is atomically incremented at the beginning of every interaction. Instead of generating the commitments μ_i via a random oracle, we can also generate it via a commitment scheme (highlighted line). As long as it is binding, one can easily verify that the proof goes through.

Theorem 7. *Let* CMT *be a randomness homomorphic commitment scheme. Further, let* $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_\lambda, \hat{\mathsf{H}} : \{0,1\}^* \to \{0,1\}^n$ *be random oracles. Then* $\mathsf{CCBS}[\mathsf{CMT}]$ *satisfies one-more unforgeability, if the* RSA *assumption holds relative to* RSAGen.

Precisely, for every adversary against the OMUF *security of* $\mathsf{CCBS}[\mathsf{CMT}]$ *that has advantage* ϵ *and makes at most* $Q_\mathsf{H}, Q_{\hat{\mathsf{H}}}$ *queries to oracles* $\mathsf{H}, \hat{\mathsf{H}}$*, respectively, starts at most* p *interactions with his signer oracle, and runs in time* t*, there exists an adversary against the binding property of* CMT *with running time* t *and success probability* ϵ_CMT *and two algorithms solving the* RSA *assumptions in time* $2t, t$ *with success probability* $\epsilon_\mathsf{RSA}, \epsilon_{\mathsf{RSA}'}$*, respectively, such that*

$$\epsilon_\mathsf{RSA} \geq \frac{1}{Q_\mathsf{H}^2 \ell^3} \left(\frac{\epsilon}{4} - \frac{\mathsf{stat}}{2} - \frac{\epsilon_\mathsf{CMT}}{2} - \frac{p\epsilon_{\mathsf{RSA}'}}{2} - \frac{(Q_\mathsf{H}(p-\ell))^{\ell+1}}{\lambda} \right)^3,$$

where $\mathsf{stat} = \left(Q_{\hat{\mathsf{H}}}^2 + pQ_{\hat{\mathsf{H}}} + p^4 + p^2 Q_\mathsf{H} \right) / 2^n$ *and* $\ell = 3\ln(p+1) + \ln(2/\epsilon)$.

H.3 Construction

Next, we present our construction BS_RSA, which makes use of a randomness homomorphic commitment scheme CMT with randomness space $\mathcal{R}_\mathsf{ck}$ and a puncturable pseudorandom function PRF. It should be mentioned that we can instantiate PRF using random oracles (cf. Supplementary Material Section E) and CMT tightly based on the RSA assumption (cf. Supplementary Material Section D.1). Furthermore, we need random oracles $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_\lambda, \mathsf{H}' : \{0,1\}^* \to \mathbb{Z}_N^*, \mathsf{H}'' : \{0,1\}^* \to \{0,1\}^n$ and $\mathsf{H}_\mathsf{r}, \mathsf{H}_c : \{0,1\}^* \to \{0,1\}^n, \mathsf{H}_x : \{0,1\}^* \to \mathcal{D} \times \mathbb{Z}_\lambda \times \mathcal{R}_\mathsf{ck} \times \{0,1\}^{n_\mathsf{PRF}}$, where n_PRF is a security parameter used for PRF.

Key Generation. Algorithm $\mathsf{BS}_\mathsf{RSA}.\mathsf{Gen}(1^n)$ generates keys as follows:

1. Generate parameters $\mathsf{par} = (N, a, \lambda)$ and a trapdoor $\mathsf{td} = (p, q)$ as above.
2. Sample $\mathsf{sk}' \leftarrow\!\!\$\, \mathcal{D}$, set $\mathsf{pk}' := \mathsf{F}(\mathsf{sk}')$.
3. Generate a commitment key $\mathsf{ck} \leftarrow \mathsf{CMT}.\mathsf{Gen}(1^n)$.
4. Set the state of S to $\mathsf{ctr} := 1$.
5. Return the public key $\mathsf{pk} := (\mathsf{par}, \mathsf{pk}', \mathsf{ck})$ and the secret key $\mathsf{sk} := (\mathsf{td}, \mathsf{sk}')$.

Signature Issuing. The algorithms S, U of the signature issuing protocol are formally presented in Figures 7 and 8. We note that S keeps a state ctr, which is inititalized as $\mathsf{ctr} := 1$.

Verification. A signature $\sigma = (c', s', \varphi^*)$ is verified with respect to a message m via algorithm $\mathsf{BS}_\mathsf{RSA}.\mathsf{Ver}(\mathsf{pk} = (\mathsf{par}, \mathsf{pk}', \mathsf{ck}), \mathsf{m}, \sigma)$, which is as follows:

1. Compute the commitment $\mu^* := \mathsf{Com}(\mathsf{ck}, \mathsf{m}; \varphi^*)$
2. Return 1 if $c' = \mathsf{H}(\mu^*, \mathsf{F}(s') \cdot \mathsf{pk}'^{-c'})$. Otherwise return 0.

$\mathsf{Check}(\mathsf{pk}, N, \mathsf{seed}, \mu_0, \mathsf{com}_\mathsf{r}, \mathsf{com}_c, J, k_J, c_J, \eta)$

1 : **for** $j \in [N] \setminus \{J\}$:
2 : $\quad \mathsf{prer}_j := \mathsf{PRF.Eval}(k_J, j),\ \mathsf{r}_j := \mathsf{H}_x(\mathsf{prer}_j)$
3 : $\quad$ **parse** $\mathsf{r}_j = (\alpha_j, \beta_j, \varphi_j, \gamma_j) \in \mathcal{D} \times \mathbb{Z}_\lambda \times \mathcal{R}_\mathsf{ck} \times \{0,1\}^{n_\mathsf{PRF}}$
4 : $\quad R_j := \mathsf{H}'(\mathsf{seed}, j)$
5 : $\quad \mu_j := \mathsf{Translate}(\mathsf{ck}, \mu_0, \varphi_j)$
6 : $\quad c_j := \mathsf{H}(\mu_j, R_j \cdot \mathsf{F}(\alpha_j) \cdot \cdot \mathsf{pk}'^{\beta_j}) + \beta_j$
7 : **if** $\mathsf{com}_\mathsf{r} \neq \mathsf{H}_\mathsf{r}(\mathsf{H}_\mathsf{r}(\mathsf{r}_1), \ldots, \mathsf{H}_\mathsf{r}(\mathsf{r}_{J-1}), \eta, \mathsf{H}_\mathsf{r}(\mathsf{r}_{J+1}), \ldots, \mathsf{H}_\mathsf{r}(\mathsf{r}_N))$: **return** 0
8 : **if** $\mathsf{com}_c \neq \mathsf{H}_c(c_1, \ldots, c_N)$: **return** 0
9 : **return** 1

Fig. 7. The algorithm Check used in the issuing protocol of blind signature scheme BS_RSA, where $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_\lambda, \mathsf{H}' : \{0,1\}^* \to \mathbb{Z}_N^*$ and $\mathsf{H}_\mathsf{r}, \mathsf{H}_c : \{0,1\}^* \to \{0,1\}^n, \mathsf{H}_x : \{0,1\}^* \to \mathcal{D} \times \mathbb{Z}_\lambda \times \mathcal{R}_\mathsf{ck} \times \{0,1\}^{n_\mathsf{PRF}}$ are random oracles.

H.4 Security Analysis

Completeness of the scheme is immediate. We show blindness and one-more unforgeability. For blindness, we will use the following lemma, stating that even with a trapdoor, the linear blind signature scheme from the OGQ linear function satisfies blindness.

Lemma 6. *For any algorithm $\mathcal{A}$ and bit $b \in \{0,1\}$, we consider the following game $\boldsymbol{G}_b$:*

1. *Let $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_\lambda$ be a random oracle. Run*
$$(\rho, \mathsf{m}_0, \mathsf{m}_1, St) \leftarrow \mathcal{A}^\mathsf{H}(1^n).$$
Use ρ as random coins to compute values $N, p, q, \lambda, a, \mathsf{sk}', \mathsf{pk}'$ as in the key generation of the scheme BS_RSA.
2. *Let $O_{b'}$ for $b' \in \{0,1\}$ be an interactive oracle. Upon termination, it locally outputs $\sigma_{b \oplus b'}$ to the game. The oracle is defined as follows:*
   (a) *Receive R from $\mathcal{A}$, sample $(\alpha, \beta) \leftarrow_s \mathcal{D} \times \mathbb{Z}_\lambda$, set $R' := R \cdot \mathsf{F}(\alpha) \cdot \mathsf{pk}'^{\beta}$. Set $c' := \mathsf{H}(\mathsf{m}_{b \oplus b'}, R')$ and $c := c' + \beta$. Send c to $\mathcal{A}$.*
   (b) *Receive s from $\mathcal{A}$. If $\mathsf{F}(s) \neq R \cdot \mathsf{pk}'^{c}$, define the local output of this oracle to be $\sigma_{b \oplus b'} := \ \perp$. Otherwise, set $s' := s \circ \alpha \circ \Psi(\mathsf{pk}', c, -c')$ and define the local output of this oracle to be $\sigma_{b \oplus b'} := (c', s')$.*
3. *Run $\mathcal{A}$ on input St with arbitrary interleaved one-time access to each of these oracles, i.e.*
$$St' \leftarrow \mathcal{A}^{O_0, O_1, \mathsf{H}}(St).$$
4. *If $\sigma_0 = \ \perp$ or $\sigma_1 = \ \perp$, run $b^* \leftarrow \mathcal{A}(St', \perp, \perp)$. Else, run $b^* \leftarrow \mathcal{A}(St', \sigma_0, \sigma_1)$. Output b^*.*

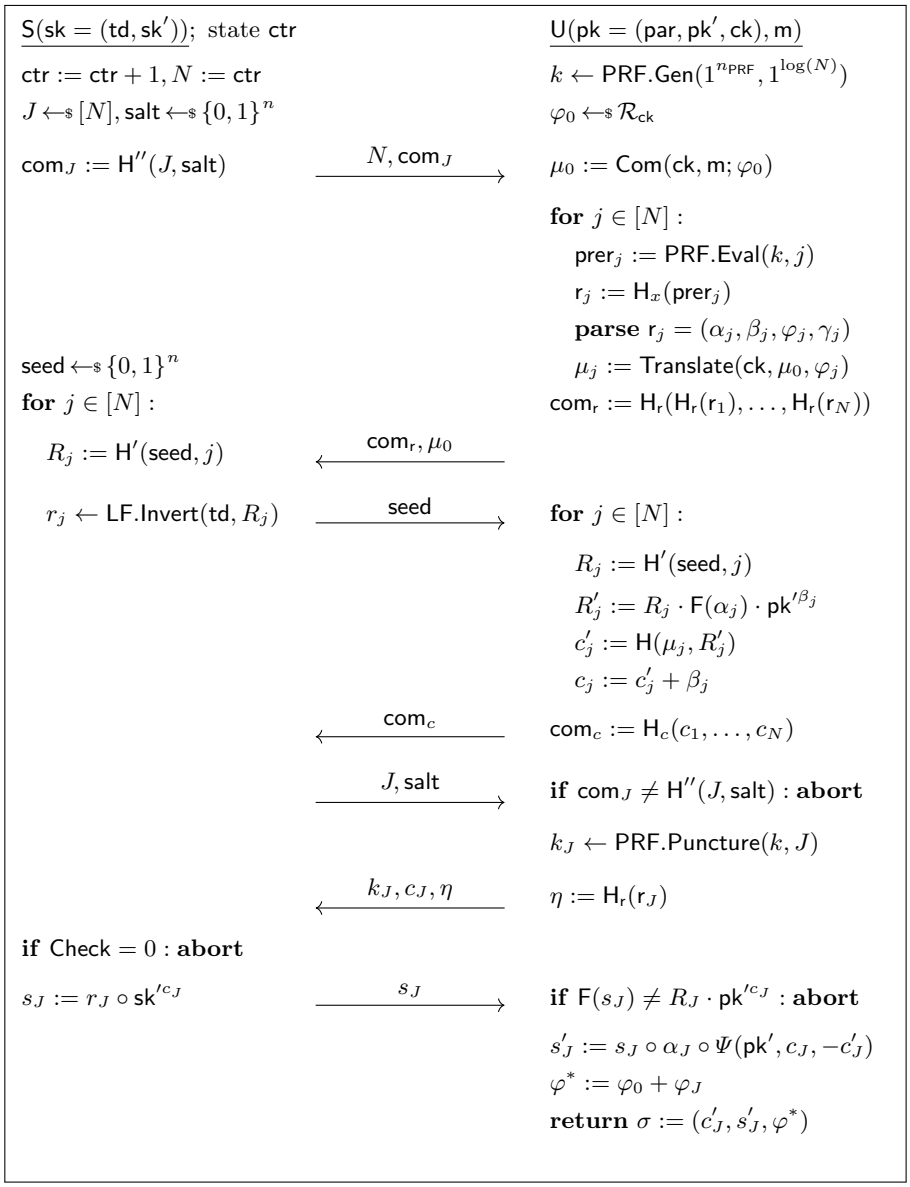

Fig. 8. The signature issuing protocol of the blind signature scheme $\mathsf{BS}_{\mathsf{RSA}}$, where $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_\lambda, \mathsf{H}' : \{0,1\}^* \to \mathbb{Z}_N^*, \mathsf{H}'' : \{0,1\}^* \to \{0,1\}^n$ and $\mathsf{H}_\mathsf{r}, \mathsf{H}_c : \{0,1\}^* \to \{0,1\}^n, \mathsf{H}_x : \{0,1\}^* \to \mathcal{D} \times \mathbb{Z}_\lambda \times \mathcal{R}_\mathsf{ck} \times \{0,1\}^{n_{\mathsf{PRF}}}$ are random oracles. The algorithm Check is defined in Figure 7. The state ctr of S is atomically incremented at the beginning of every interaction.

Then, for each algorithm $\mathcal{A}$ that makes at most Q_{H} many queries to H we have

$$|\Pr[\boldsymbol{G}_0 \Rightarrow 1] - \Pr[\boldsymbol{G}_1 \Rightarrow 1]| \leq \frac{4Q_{\mathsf{H}}}{\mathbb{Z}_N^*}.$$

Proof. This is a direct application of Theorem 6 for the OGQ linear function family. □

Theorem 8. *Let* PRF *be a puncturable pseudorandom function and* CMT *be a randomness homomorphic commitment scheme. Let* $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_\lambda, \mathsf{H}'' : \{0,1\}^* \to \{0,1\}^n$ *and* $\mathsf{H}_{\mathsf{r}} : \{0,1\}^* \to \{0,1\}^n, \mathsf{H}_x : \{0,1\}^* \to \mathcal{D} \times \mathbb{Z}_\lambda \times \mathcal{R}_{\mathsf{ck}} \times \{0,1\}^{n_{\mathsf{PRF}}}$ *be random oracles. Then* $\mathsf{BS}_{\mathsf{RSA}}$ *satisfies semi-honest signer blindness.*

In particular, for any adversary that uses N^L and N^R as the counters in its executions with the user and queries $\mathsf{H}, \mathsf{H}_{\mathsf{r}}, \mathsf{H}_x, \mathsf{H}''$ at most $Q_{\mathsf{H}}, Q_{\mathsf{H}_{\mathsf{r}}}, Q_{\mathsf{H}_x}, Q_{\mathsf{H}''}$ times, respectively, the semi-honest signer blindness advantage can be bounded by

$$4\epsilon_{\mathsf{PRF}} + \frac{Q_{\mathsf{H}''}^2}{2^{n-1}} + \frac{Q_{\mathsf{H}''}}{2^{n-2}} + \frac{Q_{\mathsf{H}_x}}{2^{n_{\mathsf{PRF}}-2}} + \frac{Q_{\mathsf{H}_{\mathsf{r}}}}{2^{n_{\mathsf{PRF}}-2}} + \frac{4Q_{\mathsf{H}}}{|\mathbb{Z}_N^*|},$$

where ϵ_{PRF} is the advantage of an adversary against the security of PRF with input length $\max\{\log(N^L), \log(N^R)\}$ puncturing at one point.

Proof. Set $\mathsf{BS} := \mathsf{BS}_{\mathsf{RSA}}$. Let $\mathcal{A}$ be a PPT algorithm and denote its advantage with respect to blindness by ϵ. In terms

$$\epsilon := \left|\Pr\left[\mathbf{BLIND}_{0,\mathsf{BS}}^{\mathcal{A}}(n) \Rightarrow 1\right] - \Pr\left[\mathbf{BLIND}_{1,\mathsf{BS}}^{\mathcal{A}}(n) \Rightarrow 1\right]\right|.$$

We will show the statement via a sequence of games. Unless otherwise stated, random oracles are simulated honestly via lazy sampling. We note that this proof is very similar to the proof of Theorem 1, except for the first two game hops.

Game $\mathbf{G}_{0,b}$: Game $\mathbf{G}_{0,b}$ is defined as the real blindness game $\mathbf{BLIND}_{b,\mathsf{BS}}^{\mathcal{A}}$. Recall that the game first obtains random coins ρ and two messages $\mathsf{m}_0, \mathsf{m}_1$ from the adversary $\mathcal{A}$. It computes a public key pk as the output of Gen on random coins ρ. Afterwards, $\mathcal{A}$ will interact with two oracles O_0 and O_1, simulating $\mathsf{U}(\mathsf{pk}, \mathsf{m}_b)$ and $\mathsf{U}(\mathsf{pk}, \mathsf{m}_{1-b})$, respectively. We will reference to the variables used in these executions using superscripts L and R, respectively. For example, J^L refers to the index J sent by $\mathcal{A}$ in the interaction with oracle O_0. If we omit the superscript, we mean that our description applies to both oracles. According to this, N^L and N^R denote the cut-and-choose parameters sent by $\mathcal{A}$ in the first message of the interaction with oracles $\mathrm{O}_0, \mathrm{O}_1$, respectively. By definition, we have

$$\epsilon = |\Pr[\mathbf{G}_{0,0} \Rightarrow 1] - \Pr[\mathbf{G}_{0,1} \Rightarrow 1]|.$$

Game $\mathbf{G}_{1,b}$: Game $\mathbf{G}_{1,b}$ is exactly as game $\mathbf{G}_{0,b}$, except that it aborts whenever there is a collision for random oracle H''. That is, whenever there are queries

$\mathsf{H}''(x) = \mathsf{H}''(x')$ for $x \neq x'$. Clearly, the distinguishing advantage between games $\mathbf{G}_{1,b}$ and $\mathbf{G}_{0,b}$ can be bounded by the probability of such a collision, which leads to

$$|\Pr[\mathbf{G}_{0,b} \Rightarrow 1] - \Pr[\mathbf{G}_{1,b} \Rightarrow 1]| \leq \frac{Q_{\mathsf{H}''}^2}{2^n}.$$

Game $\mathbf{G}_{2,b}$: Game $\mathbf{G}_{2,b}$ is exactly as game $\mathbf{G}_{1,b}$, except that we introduce another abort. In this game, whenever the adversary sends N, com_J as its first message, the game searches for a query $\mathsf{H}''(\hat{J}, \hat{\mathsf{salt}}) = \mathsf{com}_J$. Note that the game can find at most one such query due to the previous change. If the game does not find such a query, but later the user does not abort, as the adversary successfully opens $\mathsf{com}_\mathbf{J}$ by sending J, salt, the game aborts. It is easy to see that the probability of this event is at most $Q_{\mathsf{H}''}/2^n$ for fixed com_J and thus a union bound over $\{L, R\}$ leads to

$$|\Pr[\mathbf{G}_{1,b} \Rightarrow 1] - \Pr[\mathbf{G}_{2,b} \Rightarrow 1]| \leq \frac{Q_{\mathsf{H}''}}{2^{n-1}}.$$

Note that from now on, we can focus on the case where the game is able to find the query $\mathsf{H}''(\hat{J}, \hat{\mathsf{salt}}) = \mathsf{com}_J$, as otherwise the user oracle does abort. In particular, this implies that $\hat{J} = J$. If the user oracle does abort, the adversary does not learn anything about the bit b as CMT is perfectly hiding and no information about the randomness φ_0 is ever revealed to $\mathcal{A}$. For the rest of the proof, $\hat{J}$ denotes the cut-and-choose index that is extracted by the game from the commitment com_J and J is the cut-and-choose index that is later sent by $\mathcal{A}$ to open com_J. As said, we focus on the case where these are equal.

Game $\mathbf{G}_{3,b}$: Game $\mathbf{G}_{3,b}$ is defined exactly as $\mathbf{G}_{2,b}$, except that we change the way the randomness seeds prer_j are generated. Recall that before, these values were generated as in the real scheme, i.e.

$$\mathsf{prer}_j := \mathsf{PRF.Eval}(k, j) \text{ for all } j \in [N].$$

Instead, we now generate these values using a punctured key $k_{\hat{J}}$ for $j \neq \hat{J}$, and as before for $j = \hat{J}$. To be precise, at the beginning of the interaction, we sample $k \leftarrow \mathsf{PRF.Gen}(1^{n_{\mathsf{PRF}}}, 1^{\log(N)})$ as before, but additionally generate $k_{\hat{J}} \leftarrow \mathsf{PRF.Puncture}(k, \hat{J})$. Then we sample

$$\mathsf{prer}_{\hat{J}} := \mathsf{PRF.Eval}(k, \hat{J})$$

and

$$\mathsf{prer}_j := \mathsf{PRF.Eval}(k_{\hat{J}}, j) \text{ for all } j \in [N] \setminus \{\hat{J}\}.$$

Clearly, by the completeness of PRF this is only a syntactical change, and hence

$$\Pr[\mathbf{G}_{3,b} \Rightarrow 1] = \Pr[\mathbf{G}_{2,b} \Rightarrow 1].$$

Game $\mathbf{G}_{4,b}$: In game $\mathbf{G}_{4,b}$, we change the way we generate the values $\mathsf{prer}_{\hat{J}_L}^L$.

Namely, we sample $\mathsf{prer}^L_{\hat{J}^L} \leftarrow\!\!\$\, \{0,1\}^n_{\mathsf{PRF}}$. The difference between $\mathbf{G}_{3,b}$ and $\mathbf{G}_{4,b}$ can now be bounded using the security of the puncturable pseudorandom function PRF. To be precise, we construct a reduction $\mathcal{B}$ as follows:

- Run $(\rho, \mathsf{m}_0, \mathsf{m}_1, St) \leftarrow \mathcal{A}(1^n)$ and set $(\mathsf{pk}, \mathsf{sk}) := \mathsf{Gen}(1^n; \rho)$.
- Run $\mathcal{A}$ on input St with access to random oracles and interactive oracles $\mathrm{O}_0, \mathrm{O}_1$, i.e. $St' \leftarrow \mathcal{A}^{\mathrm{O}_0, \mathrm{O}_1}(St)$. The oracle O_1 is provided as in game $\mathbf{G}_{3,b}$ and oracle O_0 is provided as follows:
  - When $\mathcal{A}$ sends N^L, com^L_J, extract $\hat{J}^L$ from com^L_J as game $\mathbf{G}_{3,b}$ does and output $\hat{J}^L$ to the PRF challenger. Obtain the punctured key $k_{\hat{J}^L}$ and value $\mathsf{prer}^L_{\hat{J}^L}$.
  - Use $k_{\hat{J}^L}$ to sample prer^L_j for $j \in [N^L] \setminus \{\hat{J}^L\}$ as in $\mathbf{G}_{3,b}$. Continue the oracle simulation as in $\mathbf{G}_{3,b}$. According to this, if the simulation does not abort, send the key $k_{\hat{J}^L}$ in the sixth message of the interaction.
- Let σ_b, σ_{1-b} be the local outputs of $\mathrm{O}_0, \mathrm{O}_1$, respectively. If $\sigma_0 = \quad \perp$ or $\sigma_1 = \quad \perp$, then run $b' \leftarrow \mathcal{A}(St', \perp, \perp)$. Else, run $b' \leftarrow \mathcal{A}(St', \sigma_0, \sigma_1)$ and output b'.

Note that if $\mathsf{prer}^L_{\hat{J}^L_i}$ is random, then $\mathcal{B}$ perfectly simulates $\mathbf{G}_{4,b}$, whereas if $\mathsf{prer}^L_{\hat{J}^L_i} = \mathsf{Eval}(k, \hat{J})$, $\mathcal{B}$ perfectly simulates $\mathbf{G}_{3,b}$. By the security of PRF with input length $\log(N^L)$ we obtain

$$|\Pr[\mathbf{G}_{3,b} \Rightarrow 1] - \Pr[\mathbf{G}_{4,b} \Rightarrow 1]| \leq \epsilon_{\mathsf{PRF}}.$$

Game $\mathbf{G}_{5,b}$: In game $\mathbf{G}_{5,b}$, we change the way we generate $\mathsf{prer}^R_{\hat{J}^R}$. Namely, we sample $\mathsf{prer}^R_{\hat{J}^R} \leftarrow\!\!\$\, \{0,1\}^{n_{\mathsf{PRF}}}$. Note that we can repeat the argument we used from $\mathbf{G}_{3,b}$ to $\mathbf{G}_{4,b}$ and obtain

$$|\Pr[\mathbf{G}_{4,b} \Rightarrow 1] - \Pr[\mathbf{G}_{5,b} \Rightarrow 1]| \leq \epsilon_{\mathsf{PRF}}.$$

Game $\mathbf{G}_{6,b}$: In game $\mathbf{G}_{6,b}$, we change the way we compute $\mathsf{r}_{\hat{J}}$. Note that in $\mathbf{G}_{5,b}$ this was computed as $\mathsf{r}_{\hat{J}} := \mathsf{H}_x(\mathsf{prer}_{\hat{J}})$. Now, we sample it randomly as

$$\mathsf{r}_{\hat{J}} = (\alpha_{\hat{J}}, \beta_{\hat{J}}, \varphi_{\hat{J}}, \gamma_{\hat{J}}) \leftarrow\!\!\$\, \mathcal{D} \times \mathbb{Z}_\lambda \times \mathcal{R}_{\mathsf{ck}} \times \{0,1\}^{n_{\mathsf{PRF}}}.$$

Note that the adversary can only distinguish between games $\mathbf{G}_{5,b}$ and $\mathbf{G}_{6,b}$ if it queries $\mathsf{H}_x(\mathsf{prer}_{\hat{J}})$. However, $\mathcal{A}$ obtains no information about $\mathsf{prer}_{\hat{J}}$, which is sampled uniformly at random. By a union bound over all hash queries and $\{L, R\}$ we obtain

$$|\Pr[\mathbf{G}_{5,b} \Rightarrow 1] - \Pr[\mathbf{G}_{6,b} \Rightarrow 1]| \leq \frac{2Q_{\mathsf{H}_x}}{2^{n_{\mathsf{PRF}}}}.$$

Game $\mathbf{G}_{7,b}$: Game $\mathbf{G}_{7,b}$ is as $\mathbf{G}_{6,b}$, except that it computes $\mathsf{com}_{\mathsf{r}}$ in a different way. In detail, it samples $\eta \leftarrow\!\!\$\, \{0,1\}^n$ and computes the $\mathsf{com}_{\mathsf{r}}$ as

$$\mathsf{com}_{\mathsf{r}} := \mathsf{H}_{\mathsf{r}}(\mathsf{H}_{\mathsf{r}}(\mathsf{r}_1), \ldots, \mathsf{H}_{\mathsf{r}}(\mathsf{r}_{\hat{J}-1}), \eta, \mathsf{H}_{\mathsf{r}}(\mathsf{r}_{\hat{J}+1}), \ldots, \mathsf{H}_{\mathsf{r}}(\mathsf{r}_N))$$

Later it returns η as part of its third message. Note that $\mathcal{A}$ can only see the difference between $\mathbf{G}_{6,b}$ and $\mathbf{G}_{7,b}$ if it queries $\mathsf{H}_{\mathsf{r}}(\mathsf{r}^X_{\hat{J}X})$ for an $X \in \{L, R\}$. Note that $\mathcal{A}$ obtains no information about $\gamma_{\hat{J}}$ and $\gamma_{\hat{J}}$ is sampled uniformly at random. We can apply a union bound over all $Q_{\mathsf{H}_{\mathsf{r}}}$ random oracle queries and $X \in \{L, R\}$ and obtain

$$|\Pr[\mathbf{G}_{6,b} \Rightarrow 1] - \Pr[\mathbf{G}_{7,b} \Rightarrow 1]| \leq \frac{2Q_{\mathsf{H}_{\mathsf{r}}}}{2^{n_{\mathsf{PRF}}}}.$$

Game $\mathbf{G}_{8,b}$: In game $\mathbf{G}_{8,b}$ we change the way the commitment $\mu_{\hat{J}}$ is generated. Recall that in $\mathbf{G}_{7,b}$, this is generated as

$$\mu_{\hat{J}} := \mathsf{Translate}(\mathsf{ck}, \mu_0, \varphi_{\hat{J}}) = \mathsf{Com}(\mathsf{ck}, \mathsf{m}; \varphi_0 + \varphi_{\hat{J}}).$$

Note that if the game does not stop, then especially $\hat{J} = J$ and $\varphi^* = \varphi_0 + \varphi_{\hat{J}}$. In game $\mathbf{G}_{8,b}$, we sample $\varphi^* \leftarrow\!\!\$\, \mathcal{R}_{\mathsf{ck}}$ and set $\mu_{\hat{J}} := \mathsf{Com}(\mathsf{ck}, \mathsf{m}; \varphi^*)$. We can argue that the view of $\mathcal{A}$ is unchanged as follows. Note that due to the previous changes, $\mathcal{A}$ gets no information about $\varphi_{\hat{J}}$. Thus, we have to consider the distribution of the value $\varphi^* = \varphi_0 + \varphi_{\hat{J}}$ conditioned on $k_{\hat{J}}, (\varphi_0 + \varphi_j)_{j \neq \hat{J}}$ and φ_0. This distribution is uniformly random as $\varphi_{\hat{J}}$ is uniformly random. Hence we have

$$\Pr[\mathbf{G}_{8,b} \Rightarrow 1] = \Pr[\mathbf{G}_{7,b} \Rightarrow 1].$$

Game $\mathbf{G}_{9,b}$: In game $\mathbf{G}_{9,b}$, we change the way μ_0 is generated. In particular, we sample a random message $\bar{\mathsf{m}}^L$ (resp. $\bar{\mathsf{m}}^R$) and set $\mu_0 := \mathsf{Com}(\mathsf{ck}, \bar{\mathsf{m}}; \varphi_0)$. Note that in $\mathbf{G}_{9,b}$ the value φ_0 is only needed to compute μ_0. Especially, it is not needed to compute the value φ^* due to the previous changes. It follows from the security of CMT that $\mathsf{Com}(\mathsf{ck}, \bar{\mathsf{m}}; \varphi_0)$ and $\mathsf{Com}(\mathsf{ck}, \mathsf{m}; \varphi_0)$ are identically distributed given ck. Therefore, the view of $\mathcal{A}$ is not affected by this change and we obtain

$$\Pr[\mathbf{G}_{9,b} \Rightarrow 1] = \Pr[\mathbf{G}_{8,b} \Rightarrow 1].$$

Note that in $\mathbf{G}_{9,b}$, the only part of the oracles $\mathrm{O}_0, \mathrm{O}_1$ that depends on bit b is the $\hat{J}^{th}$ session. Also, note that each session by itself corresponds to the user algorithm of the linear blind signature scheme from the OGQ linear function family, which is statistically blind. We show this in Lemma 6. Thus, we can reduce from the games in Lemma 6 to bound $\mathcal{A}$'s advantage in distinguishing between $\mathbf{G}_{9,0}$ and $\mathbf{G}_{9,1}$. To do so, we construct a reduction $\mathcal{B}'$ as follows:

- $\mathcal{B}'$ runs $\mathcal{A}$ and gets random coins ρ and messages $\mathsf{m}_0, \mathsf{m}_1$, i.e. $(\rho, \mathsf{m}_0, \mathsf{m}_1, St) \leftarrow \mathcal{A}(1^n)$. $\mathcal{B}'$ partitions these coins into ρ_{ck} and ρ_{LF}, where ρ_{ck} are the coins that determine ck and ρ_{LF} are the coins that determine the other parts of the key, i,e. $\mathsf{par}, \mathsf{pk}'$ and $\mathsf{td}, \mathsf{sk}'$. It computes these values from the coins. Then, $\mathcal{B}'$ samples $\varphi_0^*, \varphi_1^* \leftarrow\!\!\$\, \mathcal{R}_{\mathsf{ck}}$ and sets

  $$\mu_0^* := \mathsf{Com}(\mathsf{ck}, \mathsf{m}_0; \varphi_0^*), \quad \mu_1^* := \mathsf{Com}(\mathsf{ck}, \mathsf{m}_1; \varphi_1^*).$$

  It outputs $\rho_{\mathsf{LF}}, \mu_0^*, \mu_1^*$ and its state to its challenger.

- $\mathcal{B}'$ is executed with access to oracles O'_0 and O'_1. Also, $\mathcal{B}'$ has access to a random oracle H. $\mathcal{B}'$ simulates all random oracles except H'' honestly for $\mathcal{A}$, which involves appropriately forwarding queries from $\mathcal{A}$ to its challenger for oracle H. Oracle H'' is simulated as in game $\mathbf{G}_{9,b}, b \in \{0,1\}$, i.e. it is simulated honestly but the simulation is aborted whenever a collision occurs. It runs $\mathcal{A}$ on input St with access to random oracles and interactive oracles $\mathrm{O}_0, \mathrm{O}_1$, i.e. $St' \leftarrow \mathcal{A}^{\mathrm{O}_0,\mathrm{O}_1}(St)$. We describe the simulation of oracle O_0. Oracle O_1 is simulated analogously by using O'_1 instead of O'_0:
  - When $\mathcal{A}$ sends N, com_J, extract $\hat{J}$ from com_J as $\mathbf{G}_{9,b}, b \in \{0,1\}$ does. Sample keys $k \leftarrow \mathsf{PRF.Gen}(1^{n_{\mathsf{PRF}}}, 1^{\log(N)}), k_{\hat{J}} \leftarrow \mathsf{PRF.Puncture}(k, \hat{J})$. Set $\mathsf{prer}_j := \mathsf{PRF.Eval}(k_{\hat{J}}, j)$ and set $\mathsf{r}_j := \mathsf{H}_x(\mathsf{prer}_j)$ for all $j \in [N] \setminus \{\hat{J}\}$. Sample $\varphi_0 \leftarrow\!\!\$\, \mathcal{R}_{\mathsf{ck}}$ and a random message $\bar{\mathsf{m}}$ and set $\mu_0 := \mathsf{Com}(\mathsf{ck}, \bar{\mathsf{m}}; \varphi_0)$. Set $\mu_j := \mathsf{Translate}(\mathsf{ck}, \mu_0, \varphi_j)$ for $j \in [N] \setminus \{\hat{J}\}$. . Sample $\eta \leftarrow\!\!\$\, \{0,1\}^n$ and compute $\mathsf{com}_\mathsf{r} := \mathsf{H}_\mathsf{r}(\mathsf{H}_\mathsf{r}(\mathsf{r}_1), \ldots, \mathsf{H}_\mathsf{r}(\mathsf{r}_{\hat{J}-1}), \eta, \mathsf{H}_\mathsf{r}(\mathsf{r}_{\hat{J}+1}), \ldots, \mathsf{H}_\mathsf{r}(\mathsf{r}_N))$. Send μ_0 and com_r to $\mathcal{A}$.
  - Obtain seed from adversary and compute all c_j for $j \in [N] \setminus \{\hat{J}\}$ as in the scheme using the values μ_j. For session $\hat{J}$, compute $R_{\hat{J}} := \mathsf{H}'(\mathsf{seed}, \hat{J})$ and send $R_{\hat{J}}$ to oracle O'_0. Obtain a value $c_{\hat{J}}$ and compute $\mathsf{com}_c := \mathsf{H}_c(c_1, \ldots, c_N)$. Send com_c to $\mathcal{A}$.
  - Obtain J, salt from $\mathcal{A}$. If $\mathsf{com}_J \neq \mathsf{H}''(J, \mathsf{salt})$ abort the execution of this oracle. Otherwise it must holds that $\hat{J} = J$. Continue by sending $k_{\hat{J}}, c_{\hat{J}}$ and η to $\mathcal{A}$.
  - Obtain s_J from $\mathcal{A}$ and forward it to oracle O'_0.
- $\mathcal{B}'$ obtains signatures $\sigma'_0 = (c'_0, s'_0)$ and $\sigma'_1 = (c'_1, s'_1)$ from its challenger and sets

$$\sigma_0 := (c'_0, s'_0, \varphi^*_0), \quad \sigma_1 := (c'_1, s'_1, \varphi^*_1).$$

  It runs $b' \leftarrow \mathcal{A}(St', \sigma_0, \sigma_1)$ and returns b' to the challenger.

It is easy to see that if $\mathcal{B}'$ runs in game $\mathbf{G}_0$ from Lemma 6, then it perfectly simulates game $\mathbf{G}_{9,0}$ for $\mathcal{A}$, and if it runs in game $\mathbf{G}_1$ from Lemma 6 it perfectly simulates game $\mathbf{G}_{9,1}$ for $\mathcal{A}$. Also, $\mathcal{B}'$'s output is the output of $\mathcal{A}$ and $\mathcal{B}'$ makes as many random oracle queries as $\mathcal{A}$ does (with respect to random oracle H). We know by Lemma 6 that $\mathcal{B}'$ has advantage in distinguishing the games $\mathbf{G}_0$ and $\mathbf{G}_1$ at most $4Q_\mathsf{H}/|\mathbb{Z}^*_N|$. Hence we have

$$|\Pr[\mathbf{G}_{9,0} \Rightarrow 1] - \Pr[\mathbf{G}_{9,1} \Rightarrow 1]| \leq \frac{4Q_\mathsf{H}}{|\mathbb{Z}^*_N|}.$$

The statement follows from an easy calculation. □

Theorem 9. *Let* PRF *be a puncturable pseudorandom function and* CMT *be a randomness homomorphic commitment scheme. Further, let* $\mathsf{H}' : \{0,1\}^* \to \mathbb{Z}^*_N, \mathsf{H}'' : \{0,1\}^* \to \{0,1\}^n$ *and* $\mathsf{H}_\mathsf{r}, \mathsf{H}_c : \{0,1\}^* \to \{0,1\}^n$ *be random oracles. Then* $\mathsf{BS}_{\mathsf{RSA}}$ *satisfies one-more unforgeability, assuming that* CCBS[CMT] *(cf. Supplementary Material Section H.2) does.*

Specifically, for any adversary against the OMUF *security of* $\mathsf{BS}_{\mathsf{RSA}}$ *that has advantage* ϵ *and makes at most* $Q_{\mathsf{H}_\mathsf{r}}, Q_{\mathsf{H}_c}, Q_{\mathsf{H}'}, Q_{\mathsf{H}''}, Q_\mathsf{H}$ *queries to oracles*

$\mathsf{H_r}, \mathsf{H}_c, \mathsf{H}', \mathsf{H}'', \mathsf{H}$, *respectively, and starts at most* p *interactions with his signer oracle and runs in time* t*, there exists an adversary against the* OMUF *security of* CCBS[CMT] *that makes at most* Q_{H} *queries to* H*, starts at most* p *interactions with his signer oracle, makes at most* p^2 *queries to his oracle* $\hat{\mathsf{H}}$*, runs in time* t *and has advantage* $\epsilon_{\mathsf{CCBS[CMT]}}$ *such that*

$$\epsilon \leq \frac{Q_{\mathsf{H_r}}^2}{2^n} + \frac{Q_{\mathsf{H}_c}^2}{2^n} + \frac{pQ_{\mathsf{H_r}}}{2^n} + \frac{pQ_{\mathsf{H}_c}}{2^n} + \frac{pQ_{\mathsf{H}'}}{2^n} + \frac{pQ_{\mathsf{H}''}}{2^n} + \epsilon_{\mathsf{CCBS[CMT]}}.$$

Proof. Set $\mathsf{BS} := \mathsf{BS}_{\mathsf{RSA}}$. Let $\mathcal{A}$ be an adversary against the OMUF security of BS. We denote its advantage in the one-more unforgeability game by ϵ. We prove the statement via a sequence of games. Parts of the proof are taken verbatim from the proof of Theorem 4. Fix an arbitrary polynomial ℓ.

Game $\mathbf{G}_0$: We start with game $\mathbf{G}_0 := \ell\text{-}\mathbf{OMUF}_{\mathsf{BS}}^{\mathcal{A}}$, which is the one-more unforgeability game. We briefly recall this game. First, a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^n)$ is sampled and $\mathcal{A}$ is run with concurrent access to an interactive oracle O simulating the signer $\mathsf{S}(\mathsf{sk})$. We denote the number of completed interactions between $\mathcal{A}$ and O after $\mathcal{A}$'s execution by ℓ. As we consider the random oracle model, $\mathcal{A}$ also gets access to random oracles, which are provided by the game in the standard lazy manner. When $\mathcal{A}$ finishes its execution, it outputs tuples $(\mathsf{m}_1, \sigma_1), \ldots, (\mathsf{m}_k, \sigma_k)$ and wins, if all m_i are distinct, $k > \ell$ and all signatures σ_i verify with respect to pk and m_i.

Game $\mathbf{G}_1$: This game is as $\mathbf{G}_0$, but we rule out collisions for oracles $\mathsf{H}_t, t \in \{\mathsf{r}, c\}$. To be more precise, we change the simulation of oracles $\mathsf{H}_t, t \in \{\mathsf{r}, c\}$ in the following way. If $\mathcal{A}$ queries $\mathsf{H}_t(x)$ and this value is not yet defined, the game samples an image $y \leftarrow\!\!\$\, \{0,1\}^n$. However, if there exists an $x' \neq x$ with $\mathsf{H}_t(x') = y$, the game returns $\perp$. Otherwise it behaves as before. Note that $\mathcal{A}$ can only distinguish between $\mathbf{G}_0$ and $\mathbf{G}_1$ if such a collision happens, i.e. H_t returns $\perp$. We can apply a union bound over all $Q_{\mathsf{H}_t}^2$ pairs of random oracle queries and obtain

$$|\Pr[\mathbf{G}_0 \Rightarrow 1] - \Pr[\mathbf{G}_1 \Rightarrow 1]| \leq \frac{Q_{\mathsf{H_r}}^2}{2^n} + \frac{Q_{\mathsf{H}_c}^2}{2^n}.$$

In particular, the previous change implies that at each point of the execution of the game and for each image $y \in \{0,1\}^n$, there is at most one preimage $\mathsf{H}_t^{-1}(y)$ under H_t. Thus, from a given image $y \in \{0,1\}^n$, the game can extract at most one preimage $x \in \{0,1\}^*$ such that $\mathsf{H}_t(x) = y$. We will use this in the following games.

Game $\mathbf{G}_2$: In game $\mathbf{G}_2$, we change the way the signing oracle is executed. Whenever $\mathcal{A}$ sends $\mathsf{com_r}, \mu_0$ as its first message, the game tries to extract the messages from this commitment. To do so, the game goes through all the random oracle queries to $\mathsf{H_r}$ and tries to find a preimage of $\mathsf{com_r}$. Then, it parses this bitstring as N hashes $h_1, \ldots, h_N$ and tries to find preimages of these in the same

way. As a result of this extraction, the game will end up with values $\bar{\mathsf{r}}_1, \ldots, \bar{\mathsf{r}}_N$, where we write $\bar{\mathsf{r}}_j = \ \perp$ if the game was not able to extract the j^{th} value. If there is a session $j \in [N]$ such that $\bar{\mathsf{r}}_j = \ \perp$, i.e. the game could not extract the randomness for that session, and later in that execution $J \neq j$ but algorithm Check outputs 1, the game aborts. Denote this event by bad_1. The probability of bad_1 is an upper bound on the distinguishing advantage of $\mathcal{A}$ between $\mathbf{G}_1$ and $\mathbf{G}_2$. For each fixed interaction, we can bound the probability of this event (with respect to that interaction) by $Q_{\mathsf{H_r}}/2^n$. To see this, consider the case where the game could not extract the values $h_1, \ldots, h_N$. Then, once $\mathsf{com_r}$ is fixed, the probability that one of the hash queries of $\mathcal{A}$ evaluates to $\mathsf{com_r}$ is at most $1/2^n$. Similarly, in the case where the game could extract h_j but could not extract a value $\bar{\mathsf{r}}_j$ from h_j for $J \neq j$, the probability that one of the hash queries of $\mathcal{A}$ evaluates to the fixed h_j is at most $1/2^n$. By a union bound over all interactions we obtain

$$|\Pr[\mathbf{G}_1 \Rightarrow 1] - \Pr[\mathbf{G}_2 \Rightarrow 1]| \leq \Pr[\mathsf{bad}_1] \leq \frac{pQ_{\mathsf{H_r}}}{2^n}.$$

Game $\mathbf{G}_3$: Again, we change the signing oracle by introducing an additional abort. Namely, whenever the adversary sends the commitment com_c as its second message, the game extracts values $\bar{c}_1, \ldots, \bar{c}_N$ such that $\mathsf{H}_c(\bar{c}_1, \ldots, \bar{c}_N) = \mathsf{com}_c$ by going through all random oracle queries of $\mathcal{A}$. If the game is not able to extract, but later algorithm Check outputs 1, the game aborts and we say that the event bad_2 occurs. Note that algorithm Check internally checks if

$$\mathsf{com}_c \neq \mathsf{H}_c(c_1, \ldots, c_N).$$

Thus, the probability of bad_2 in a fixed interaction and hence the distinguishing advantage of $\mathcal{A}$ between $\mathbf{G}_2$ and $\mathbf{G}_3$ is bounded by $Q_{\mathsf{H}_c}/2^n$, using a similar argument as in the previous game. We obtain

$$|\Pr[\mathbf{G}_2 \Rightarrow 1] - \Pr[\mathbf{G}_3 \Rightarrow 1]| \leq \Pr[\mathsf{bad}_2] \leq \frac{pQ_{\mathsf{H}_c}}{2^n}.$$

Game $\mathbf{G}_4$: This game aborts whenever the following bad event occurs. The event is defined as follows: The game samples seed after $\mathcal{A}$ sends its first message of an interaction with the signer oracle and at this point there exists an index $j \in [N]$ such that $\mathsf{H}'(\mathsf{seed}, j)$ is already defined. As seed is sampled uniformly at random from $\{0,1\}^n$ and hidden from $\mathcal{A}$ until the point of the potential abort, a union bound over all hash queries and interactions shows that

$$|\Pr[\mathbf{G}_3 \Rightarrow 1] - \Pr[\mathbf{G}_4 \Rightarrow 1]| \leq \frac{pQ_{\mathsf{H}'}}{2^n}.$$

Game $\mathbf{G}_5$: In $\mathbf{G}_5$, the signer oracle sends a random com_J in the beginning of each interaction. Later, before it has to send J, salt, it samples $J \leftarrow\!\!\$\, [N]$ and $\mathsf{salt} \leftarrow\!\!\$\, \{0,1\}^n$ and aborts if $\mathsf{H}''(J, \mathsf{salt})$ is already defined. If it is not yet defined, it defines it as $\mathsf{H}''(J, \mathsf{salt}) := \mathsf{com}_J$. The adversary $\mathcal{A}$ can only distinguish between

$\mathbf{G}_4$ and $\mathbf{G}_5$ if $\mathsf{H}''(J, \mathsf{salt})$ is already defined. By a union bound over all $Q_{\mathsf{H}''}$ hash queries and p interactions we obtain

$$|\Pr[\mathbf{G}_4 \Rightarrow 1] - \Pr[\mathbf{G}_5 \Rightarrow 1]| \leq \frac{pQ_{\mathsf{H}''}}{2^n}.$$

Let us summarize what we have so far. We changed the game step by step and ruled out the following types of bad events:

1. The adversary sends some commitment for which the game can not extract some of the committed values, but later the adversary can open it successfully.
2. The game samples a random seed such that the random oracle values of interest are already defined for that seed.

In particular, from the first property we can derive that whenever the game does not abort, it could successfully extract values all of the values $\bar{c}_1, \ldots, \bar{c}_N$. Additionally, we know by the collision freeness of H_c that we must have $c_j = \bar{c}_j$ for all $j \in [N]$. A similar statement holds for the $\bar{\mathsf{r}}_j$. Here, it can only be the case that the game can not extract a single $\bar{\mathsf{r}}_j$ but later $J = j$. On the other hand, the second property tells us that a potential reduction simulating $\mathbf{G}_5$ can program the random oracle before sending the seed or the cut-and-choose index J to the adversary. We will use these properties to construct a (tight) reduction $\mathcal{B}$ that breaks the one-more unforgeability of $\mathsf{CCBS}[\mathsf{CMT}]$ whenever $\mathbf{G}_5$ outputs 1. Reduction $\mathcal{B}$ works as follows:

- $\mathcal{B}$ gets as input $\mathsf{pk} = (\mathsf{par}, \mathsf{pk}', \mathsf{ck})$ and oracle access to a signer oracle $\hat{\mathrm{O}}$ and random oracles $\mathsf{H}, \hat{\mathsf{H}}$ for blind signature scheme $\mathsf{CCBS}[\mathsf{CMT}]$. It runs $\mathcal{A}$ with input pk and oracle access to random oracles $\mathsf{H}, \mathsf{H}', \mathsf{H}'', \mathsf{H}_\mathsf{r}$ and H_c and a signer oracle O. The oracles $\mathsf{H}', \mathsf{H}''$ are simulated honestly by $\mathcal{B}$ and oracles $\mathsf{H}_\mathsf{r}, \mathsf{H}_c$ are simulated exactly as in game $\mathbf{G}_5$.
- When adversary $\mathcal{A}$ queries oracle O to start an interaction, the reduction $\mathcal{B}$ behaves as follows:
  - It starts an interaction with oracle $\hat{\mathrm{O}}$ and obtains a parameter N as the first message. It forwards N, com_J to $\mathcal{A}$, where $\mathsf{com}_J \leftarrow\!\!\$ \{0,1\}^n$.
  - When $\mathcal{A}$ sends its first message $\mathsf{com}_\mathsf{r}, \mu_0$, $\mathcal{B}$ extracts $(\bar{\mathsf{r}}_1, \ldots, \bar{\mathsf{r}}_N)$ as in game $\mathbf{G}_5$ (cf. $\mathbf{G}_2$). For each such $j \in [N]$ for which $\bar{\mathsf{r}}_j$ is defined, it parses $\bar{\mathsf{r}}_j = (\bar{\alpha}_j, \bar{\beta}_j, \bar{\varphi}_j, \bar{\gamma}_j)$ and sets $\bar{\mu}_j := \mathsf{Translate}(\mathsf{ck}, \mu_0, \bar{\varphi}_j)$. Then it defines $\mathsf{com}_j := \hat{\mathsf{H}}(\bar{\alpha}_j, \bar{\beta}_j, \bar{\mu}_j, \bar{\gamma}_j)$. For the remaining j, it samples $\mathsf{com}_j \leftarrow\!\!\$ \{0,1\}^n$. Finally, it sends $\mathsf{com}_1, \ldots, \mathsf{com}_N$ to its oracle $\hat{\mathrm{O}}$.
  - The oracle $\hat{\mathrm{O}}$ returns $R_1, \ldots, R_N$. Then, $\mathcal{B}$ samples $\mathsf{seed} \leftarrow\!\!\$ \{0,1\}^n$. It aborts, if there exists an index $j \in [N]$ such that $\mathsf{H}'(\mathsf{seed}, j)$ is already defined. Otherwise, it programs $\mathsf{H}'(\mathsf{seed}_R, j) := R_j$ for all $j \in [N]$ and sends seed to $\mathcal{A}$.
  - When $\mathcal{A}$ sends its second message com_c, the game extracts values $\bar{c}_i$ from com_c. If this extraction is successful, i.e. $\bar{c}_j$ is defined, it sets $c'_j := \bar{c}_j$. Otherwise, it sets $c'_j \leftarrow\!\!\$ \mathcal{S}$. It sends $c'_1, \ldots, c'_N$ to $\hat{\mathrm{O}}$.

- The oracle $\hat{\mathrm{O}}$ returns an index $J \in [N]$, whereupon reduction $\mathcal{B}$ samples $\mathsf{salt} \leftarrow\!\!\$\, \{0,1\}^n$ and aborts if $\mathsf{H}''(J, \mathsf{salt})$ is already defined. Otherwise it sets $\mathsf{H}''(J, \mathsf{salt}) := \mathsf{com}_J$ and sends J, salt to $\mathcal{A}$.
- When adversary sends its third message k_J, c_J, η, algorithm $\mathcal{B}$ runs algorithm Check. If this algorithm returns 0, $\mathcal{B}$ aborts this interaction. If it outputs 1 it aborts the entire execution if one of the following two conditions hold
  - There is some index $j \in [N]$ such that $c_j = \ \perp$.
  - There is some index $j \in [N]$ such that $j \neq J$ and $\bar{\mathsf{r}}_j = \ \perp$.

  Otherwise, $\mathcal{B}$ sends $\{(\bar{\alpha}_j, \bar{\beta}_j, \bar{\mu}_j, \bar{\gamma}_j)\}_{j \neq J}$ to $\hat{\mathrm{O}}$. Note that these values are defined by the second condition that has been checked before.
- The oracle $\hat{\mathrm{O}}$ returns s_J and $\mathcal{B}$ forwards it to $\mathcal{A}$.

– When $\mathcal{A}$ outputs $(\mathsf{m}_1, \sigma_1), \ldots, (\mathsf{m}_k, \sigma_k)$, $\mathcal{B}$ outputs $(\mathsf{m}_1, \sigma_1), \ldots, (\mathsf{m}_k, \sigma_k)$.

It is easy to see that the values $R_1, \ldots, R_N$ are distributed uniformly over $\mathbb{Z}_N^*$ [9] and therefore the programming of the random oracle H' is done correctly. Further, we claim that whenever $\mathcal{B}$ does not abort during the interaction, the signing oracle $\hat{\mathrm{O}}$ will also not abort. From this claim it follows that the simulation provided by $\mathcal{B}$ is perfect. To see that the claim is true, note that $\hat{\mathrm{O}}$ could abort the signing interaction for two reasons: First, it may abort as there exists some $j \in [N]$ such that $j \neq J$ and $\mathsf{com}_j \neq \hat{\mathsf{H}}(\bar{\alpha}_j, \bar{\beta}_j, \bar{\mu}_j, \bar{\gamma}_j)$. However, this can not happen due to the way $\mathcal{B}$ defines com_j. The second reason for an abort is that there exists a $j \in [N]$ such that $j \neq J$ and $c'_j \neq \mathsf{H}(\bar{\mu}_j, R_j \cdot \mathsf{F}(\bar{\alpha}_j) \cdot \mathsf{pk}'^{\bar{\beta}_j}) + \bar{\beta}_j$. However, as we already noticed above, if $\mathbf{G}_6$ does not abort, then we have $c'_j = c_j, \bar{\mu}_j = \mu_j, \bar{\alpha}_j = \alpha_j$ and $\bar{\beta}_j = \beta_j$ and thus the $\mathcal{B}$ itself would have aborted as Check returns 0. Finally, it is clear that $\mathcal{B}$ wins the one-more unforgeability game whenever $\mathbf{G}_6$ outputs 1, as $\mathcal{B}$ outputs $\mathcal{A}$'s output and completes as many interactions with oracle $\hat{\mathrm{O}}$ as $\mathcal{A}$ completes with O. The statement follows by an easy calculation. □

H.5 Concrete Parameters and Efficiency

To derive concrete parameters for our scheme $\mathsf{BS}_{\mathsf{RSA}}$ based on the RSA assumption in a theoretically sound way, we recall the concrete security bounds from Theorems 7 and 9. Let ϵ, t denote the advantage and running time of an adversary against the one-more unforgeability of $\mathsf{BS}_{\mathsf{RSA}}$ initiating at most p interactions with the signing oracle and querying the random oracle at most Q_{H} many times. Then there is an adversary against the one-more unforgeability $\mathsf{CCBS}[\mathsf{CMT}]$ with advantage ϵ_{CCBS} and running time t. Also, there are three algorithms solving two instances of the RSA problem with probability $\epsilon_{\mathsf{RSA}}, \epsilon_{\mathsf{RSA}'}, \epsilon_{\mathsf{RSA}''}$ and running time $2t, t, t$, respectively. Here, the third adversary against RSA comes from the binding property of the commitment scheme we use.

Concretely, by combining the concrete bounds given in Theorems 7 and 9 we obtain that

$$\epsilon \leq 2\sqrt[3]{Q_{\mathsf{H}}^2 \ell^3 \epsilon_{\mathsf{RSA}}} + \frac{(Q_{\mathsf{H}}(p-\ell))^{\ell+1}}{\lambda} + 2\epsilon_{\mathsf{RSA}''} + 2p\epsilon_{\mathsf{RSA}'} + 2T_1 + T_2,$$

[9] This property is called smoothness in [25].

where T_1, T_2 are statistically negligible terms and $\ell = 3\ln(p+1) + \ln(2/\epsilon_{\mathsf{CCBS}})$. To simplify further, we assume κ bit of security for the instance related to ϵ_{RSA} and $\epsilon_{\mathsf{RSA}'}$ and κ'' bit of security for the instance related to $\epsilon_{\mathsf{RSA}''}$. By definition, this means that

$$\epsilon_{\mathsf{RSA}} < 2 \cdot t \cdot 2^{-\kappa},\ \epsilon_{\mathsf{RSA}'} < t \cdot 2^{-\kappa},\ \epsilon_{\mathsf{RSA}''} < t \cdot 2^{-\kappa''}.$$

Next, we use

$$\ell = 3\ln(p+1) + \ln\left(\frac{2}{\epsilon_{\mathsf{CCBS}}}\right) \leq 3\ln(p+1) + \ln\left(\frac{2}{\epsilon - T_2}\right) =: \ell_\epsilon.$$

Pluggin in, we get

$$\epsilon \leq 2\sqrt[3]{Q_{\mathsf{H}}^2 \ell_\epsilon^3 \cdot 2 \cdot t \cdot 2^{-\kappa}} + \frac{(Q_{\mathsf{H}} p)^{\ell_\epsilon + 1}}{\lambda} + 2t \cdot 2^{-\kappa''} + 2pt \cdot 2^{-\kappa} + 2T_1 + T_2,$$

which must hold for any adversary with running time t and advantage ϵ and any λ we choose. Note that we can set the bitlength of the prime λ independently of the RSA modulus length.

To get k bit of security for $\mathsf{BS}_{\mathsf{RSA}}$, we consider any fix choice of ϵ, t such that $t/\epsilon = 2^k$ and increase κ, κ'' until the above inequality leads to a contradiction. Then, we choose the maximum values for κ, κ''. We note that we have to take this two-step approach and iterate over all combinations of ϵ, t, as ℓ_ϵ depends on ϵ which leads to a non-linear inequality. Also, we note that we can set κ'' to be much less than κ as the relation between k and κ'' is tight. Once the appropriate security levels κ and κ'' are found, we determine the modulus lengths $\mathsf{len}, \mathsf{len}''$ following an estimation for the sub-exponential complexity of the general number field sieve algorithm [11], which is similar to [25]. Using the modulus length and the bitlength of λ, we can compute the sizes of signatures and keys.

Next, we consider blindness. For simplicity, assume that $N^L = N^R =: N$. Also, we can make the assumption[10] that $|\mathbb{Z}_N^*| \geq 2^n$. If we want to achieve blindness with k bits of security, we have to make sure that the blindness advantage is at most $2^{-k} \cdot t$. As for our CDH-based scheme, we instantiate PRF using the GGM construction (cf. Supplementary Material Section E). Using Theorem 8, the blindness advantage can be upper bounded by

$$\frac{(2\log(N) - 1)Q_{\mathsf{H}_{\mathsf{PRF}}}}{2^{n_{\mathsf{PRF}}-2}} + \frac{Q_{\mathsf{H}''}^2}{2^{n-1}} + \frac{Q_{\mathsf{H}''}}{2^{n-2}} + \frac{Q_{\mathsf{H}_x}}{2^{n_{\mathsf{PRF}}-2}} + \frac{Q_{\mathsf{H}_\mathsf{r}}}{2^{n_{\mathsf{PRF}}-2}} + \frac{Q_{\mathsf{H}}}{2^{n-2}}.$$

Thus, we only have to choose n_{PRF} large enough.

We implemented the approach in a simple Python script (cf. Supplementary Material Section K.1). Example instantiations of our parameters can be found in Table 1.

[10] This is without loss of generality, as we have to choose a security level larger than n for the underlying RSA levels.

I Ommitted Analysis of Our Scheme from CDH

Here, we give the remaining parts of the formal analysis of our scheme from CDH. In particular, we show blindness. To do so, we first introduce a lemma that will be useful for our blindness proof.

Lemma 7. *For any algorithm $\mathcal{A}$, parameters $\mathsf{par} := (\mathbb{G}, g, p, e) \leftarrow \mathsf{PGGen}(1^n)$, and bit $b \in \{0,1\}$, we consider the following game $\boldsymbol{G}_b$:*

1. *Let $\mathsf{H} : \{0,1\}^* \to \mathbb{G}$[11]. Run $((\mathsf{pk}_i, \mathsf{m}_{i,0}, \mathsf{m}_{i,1})_{i\in[K]}, St) \leftarrow \mathcal{A}^{\mathsf{H}}(\mathsf{par})$.*
2. *Let $O_{b'}$ for $b' \in \{0,1\}$ be an interactive oracle. Upon termination, it locally outputs $\sigma_{b\oplus b'}$ to the game. The oracle is defined as follows:*
   (a) *Upon a query from $\mathcal{A}$, sample $\alpha_i \leftarrow_s \mathbb{Z}_p$ and set $c_i := \mathsf{H}(\mathsf{pk}_i, \mathsf{m}_{i,b\oplus b'}) \cdot g^{\alpha_i}$ for all $i \in [K]$. Send $c_1, \ldots, c_K$ to $\mathcal{A}$.*
   (b) *Receive $\bar{s}$ from $\mathcal{A}$ and set*

$$\bar{\sigma} := \bar{s} \cdot \prod_{i=1}^{K} \mathsf{pk}_i^{-\alpha_i}.$$

   If

$$e(\bar{\sigma}, g) \neq \prod_{i\in[K]} e(\mathsf{H}(\mathsf{pk}_i, \mathsf{m}_{i,b\oplus b'}), \mathsf{pk}_i),$$

   define the local output of this oracle to be $\sigma_{b\oplus b'} := \ \perp$. Otherwise, define the local output of this oracle to be $\sigma_{b\oplus b'} := \bar{\sigma}$.
3. *Run $\mathcal{A}$ on input St with arbitrary interleaved one-time access to each of these oracles, i.e.*

$$St' \leftarrow \mathcal{A}^{O_0, O_1, \mathsf{H}}(St).$$

4. *If $\sigma_0 = \ \perp$ or $\sigma_1 = \ \perp$, run $b^* \leftarrow \mathcal{A}(St', \perp, \perp)$. Else, run $b^* \leftarrow \mathcal{A}(St', \sigma_0, \sigma_1)$. Output b^*.*

Then, for each algorithm $\mathcal{A}$, we have $\Pr[\boldsymbol{G}_0 \Rightarrow 1] = \Pr[\boldsymbol{G}_1 \Rightarrow 1]$.

Proof. We show the claim via a statistical argument. To this end, recall that the exponentiation map $\mathbb{Z}_p \to \mathbb{G}, x \mapsto g^x$ is a bijection. Thus, for each public key pk_i output by $\mathcal{A}$, we can write $\mathsf{pk}_i = g^{\mathsf{sk}_i}$. For now, we denote the discrete logarithm of an element $h \in \mathbb{G}$ with respect to g by $\mathsf{dlog}(h)$. With this notation, we have $\mathsf{sk}_i = \mathsf{dlog}(\mathsf{pk}_i)$ for all $i \in [K]$. After the adversary outputs $(\mathsf{pk}_i, \mathsf{m}_{i,0}, \mathsf{m}_{i,1})_{i\in[K]}$, we consider the rest of the experiment in two phases.

First, we consider the view of $\mathcal{A}$ before it receives σ_0 and σ_1. Here, it is clear that $\mathcal{A}$'s view in $\mathbf{G}_b$ is the same for both $b = 0$ and $b = 1$. Indeed, in both games, $\mathcal{A}$ obtains from both oracles K independent and uniform group elements c_i, as the values α_i act as a one-time pad, hiding $\mathsf{H}(\mathsf{pk}_i, \mathsf{m}_{i,b\oplus b'})$ and thus b.

[11] We do not need to model H as a random oracle here.

Next, we consider the view of $\mathcal{A}$ after it receives σ_0 and σ_1. If both are $\perp$, then $\mathcal{A}$ obtains no new information about b. On the other hand, if $\sigma_0 \neq \perp$ and $\sigma_1 = \perp$ we know that, by definition of game $\mathbf{G}_b$, we have

$$\begin{aligned} & e(\sigma_j, g) = \prod_{i\in[K]} e(\mathsf{H}(\mathsf{pk}_i, \mathsf{m}_{i,j}), \mathsf{pk}_i) \\ \iff \quad & e(g,g)^{\mathsf{dlog}(\sigma_j)} = e(g,g)^{\sum_{i\in[K]} \mathsf{dlog}(\mathsf{H}(\mathsf{pk}_i,\mathsf{m}_{i,j}))\mathsf{sk}_i} \\ \iff \quad & \sigma_j = \prod_{i\in[K]} \mathsf{H}(\mathsf{pk}_i, \mathsf{m}_{i,j})^{\mathsf{sk}_i}. \end{aligned}$$

for each $j \in \{0,1\}$, where the last equivalence follows from the non-degeneracy of the pairing. Thus, for each $j \in \{0,1\}$, the element σ_j is completely determined by $(\mathsf{pk}_i, \mathsf{m}_{i,j})_{i\in[K]}$. This implies that after learning σ_0 and σ_1, $\mathcal{A}$ obtains no additional information about bit b. Therefore, the claim follows. □

Proof (of Theorem 3). This can be proven in an analogous way to Theorem 8. The only difference is that we puncture the key at K points (one per instance) and apply the perfect blindness of the underlying blind signature scheme [4,5] K times.

We now present the details. Let $\mathsf{BS} := \mathsf{PIKA}_{\mathsf{CDH}}$ and $\mathcal{A}$ be an algorithm with blindness advantage ϵ. That is,

$$\epsilon := \left| \Pr\left[\mathbf{BLIND}^{\mathcal{A}}_{0,\mathsf{BS}}(n) \Rightarrow 1\right] - \Pr\left[\mathbf{BLIND}^{\mathcal{A}}_{1,\mathsf{BS}}(n) \Rightarrow 1\right] \right|.$$

We show that claimed uppor bound on ϵ via a sequence of games, where all random oracles are simulated honestly via lazy sampling unless otherwise specified.

Game $\mathbf{G}_{0,b}$: Game $\mathbf{G}_{0,b}$ is defined as the real blindness game $\mathbf{BLIND}^{\mathcal{A}}_{b,\mathsf{BS}}$. Let us recall this game. First, $\mathcal{A}$ outputs a public key pk and messages $\mathsf{m}_0, \mathsf{m}_1$. Then, the game provides two interactive oracles $\mathrm{O}_0, \mathrm{O}_1$ to $\mathcal{A}$, which simulate the user algorithm $\mathsf{U}(\mathsf{pk}, \mathsf{m}_b), \mathsf{U}(\mathsf{pk}, \mathsf{m}_{1-b})$, respectively. Throughout the proof, we will reference to the variables used in these executions using superscripts L and R, respectively. For example, $\mathsf{com}^L_{\mathbf{J}}$ refers to the commitment on the seed of the cut-and-choose index sent by $\mathcal{A}$ as part of the first message in the interaction with oracle O_0. If we omit the superscript, our description applies to both oracles. According to this, N^L and N^R denote the cut-and-choose parameters sent by $\mathcal{A}$ in the first message of the interaction with oracles $\mathrm{O}_0, \mathrm{O}_1$, respectively. It follows that

$$\epsilon = \left| \Pr\left[\mathbf{G}_{0,0} \Rightarrow 1\right] - \Pr\left[\mathbf{G}_{0,1} \Rightarrow 1\right] \right|.$$

Game $\mathbf{G}_{1,b}$: Game $\mathbf{G}_{1,b}$ is exactly as game $\mathbf{G}_{0,b}$, but whenever there are queries $\mathsf{H}'(x) = \mathsf{H}'(x')$ for $x \neq x'$, the game aborts. Clearly, the probability of such a collision is at most $Q^2_{\mathsf{H}'}/2^n$, which leads to

$$\left| \Pr\left[\mathbf{G}_{0,b} \Rightarrow 1\right] - \Pr\left[\mathbf{G}_{1,b} \Rightarrow 1\right] \right| \leq \frac{Q^2_{\mathsf{H}'}}{2^n}.$$

Game $\mathbf{G}_{2,b}$: Game $\mathbf{G}_{2,b}$ is exactly as game $\mathbf{G}_{1,b}$, but we introduce another abort. Namely, the game aborts if the adversary sends $N, \mathsf{com}_{\mathbf{J}}$ as its first message, but at that point the game can not find a query $\mathsf{H}'(\hat{\mathsf{seed}}_{\mathbf{J}}, \hat{\mathsf{salt}}) = \mathsf{com}_{\mathbf{J}}$ and later the user algorithm does not abort, i.e. $\mathcal{A}$ is able to successfully open $\mathsf{com}_{\mathbf{J}}$ by sending $\mathsf{seed}_{\mathbf{J}}, \mathsf{salt}$. Note that there is at most one query that the game can find, as we ruled out collisions for oracle H' in the previous change. Clearly, the probability that the adversary can successfully open a commitment for which the game can not find a query is at most $Q_{\mathsf{H}'}/2^n$. A union bound over oracles O_0 and O_1 shows that

$$|\Pr[\mathbf{G}_{1,b} \Rightarrow 1] - \Pr[\mathbf{G}_{2,b} \Rightarrow 1]| \leq \frac{Q_{\mathsf{H}'}}{2^{n-1}}.$$

Note that if the user oracle aborts, then the adversary gets $(\perp, \perp)$ in the end of the game and learns nothing about the bit b as CMT is perfectly hiding and no information about the randomness φ_0 is ever revealed to $\mathcal{A}$. Thus, from now on, we can focus on the case where the user oracle does not abort. By the change we introduced here, we can thus assume that the game is able to extract $\hat{\mathsf{seed}}_{\mathbf{J}}$ from $\mathsf{com}_{\mathbf{J}}$ and that later $\hat{\mathsf{seed}}_{\mathbf{J}} = \mathsf{seed}_{\mathbf{J}}$. For the extracted seed $\hat{\mathsf{seed}}_{\mathbf{J}}$, we also define the cut-and-choose vector $\hat{\mathbf{J}}$ and the set $\hat{\mathcal{J}}$ as

$$\forall i \in [K] : \hat{\mathbf{J}}_i := \mathsf{H}'(\hat{\mathsf{seed}}_{\mathbf{J}}, i), \quad \hat{\mathbf{J}} = (\hat{\mathbf{J}}_1, \ldots, \hat{\mathbf{J}}_K), \quad \hat{\mathcal{J}} := \{(i, \hat{\mathbf{J}}_i) \mid i \in [K]\}.$$

As $\hat{\mathsf{seed}}_{\mathbf{J}} = \mathsf{seed}_{\mathbf{J}}$, we also have $\hat{\mathbf{J}} = \mathbf{J}$ and $\hat{\mathcal{J}} = \mathcal{J}$.

Game $\mathbf{G}_{3,b}$: Game $\mathbf{G}_{3,b}$ is defined exactly as $\mathbf{G}_{2,b}$, except that we change the way the randomness seeds $\mathsf{prer}_{i,j}$ are generated. We recall that in previous games, these values were generated as in the real scheme, i.e.

$$\mathsf{prer}_{i,j} := \mathsf{PRF.Eval}(k, (i,j)) \text{ for all } (i,j) \in [K] \times [N].$$

Instead, we now generate these values using a punctured key $k_{\hat{\mathbf{J}}}$ for $(i,j) \in [K] \times [N] \setminus \hat{\mathcal{J}}$, and as before for $(i,j) \in \hat{\mathcal{J}}$. Concretely, at the beginning of the interaction, the game samples $k \leftarrow \mathsf{PRF.Gen}(1^{n_{\mathsf{PRF}}}, 1^{\log(KN)})$ as before, extracts $\hat{\mathsf{seed}}_{\mathbf{J}}$ and computes $\hat{\mathcal{J}}$ as described in $\mathbf{G}_{2,b}$, and additionally generates $k_{\hat{\mathbf{J}}} \leftarrow \mathsf{PRF.Puncture}(k, \hat{\mathcal{J}})$. Then it sets

$$\mathsf{prer}_{i,\hat{\mathbf{J}}_i} := \mathsf{PRF.Eval}(k, (i, \hat{\mathbf{J}}_i)) \text{ for all } i \in [K]$$

and

$$\mathsf{prer}_{i,j} := \mathsf{PRF.Eval}(k_{\hat{\mathbf{J}}}, (i,j)) \text{ for all } (i,j) \in [K] \times [N] \setminus \hat{\mathcal{J}}.$$

By the completeness of PRF this is only a syntactical change, and hence

$$\Pr[\mathbf{G}_{3,b} \Rightarrow 1] = \Pr[\mathbf{G}_{2,b} \Rightarrow 1].$$

Game $\mathbf{G}_{4,b}$: In game $\mathbf{G}_{4,b}$, we change the way we generate the randomness seeds $\mathsf{prer}^L_{i,\hat{\mathbf{J}}^L_i}$ for $i \in [K]$. Concretely, we sample them at random from $\{0,1\}^n_{\mathsf{PRF}}$. We can bound the distinguishing advantage between games $\mathbf{G}_{3,b}$ and $\mathbf{G}_{4,b}$ using a reduction $\mathcal{B}$ from the security of PRF. The reduction $\mathcal{B}$ is as follows:

- Run $\mathcal{A}$ to get a public key and messages, i.e. $(\mathsf{pk}, \mathsf{m}_0, \mathsf{m}_1, St) \leftarrow \mathcal{A}(1^n)$.
- Run $\mathcal{A}$ on input St with access to random oracles and interactive oracles $\mathrm{O}_0, \mathrm{O}_1$, i.e. $St' \leftarrow \mathcal{A}^{\mathrm{O}_0, \mathrm{O}_1}(St)$. The oracle O_1 is provided as in game $\mathbf{G}_{3,b}$ and oracle O_0 is provided as follows:
  - When $\mathcal{A}$ sends $N^L, \mathsf{com}_{\mathbf{J}}^L$, extract $\hat{\mathbf{J}}^L, \hat{\mathcal{J}}^L$ from com_J^L as game $\mathbf{G}_{3,b}$ does and output $\hat{\mathcal{J}}^L$ to the PRF challenger. Obtain the punctured key $k_{\hat{\mathbf{J}}^L}$ and values $\{\mathsf{prer}^L_{i,\hat{\mathbf{J}}^L_i}\}_{i\in[K]}$.
  - Use $k_{\hat{\mathbf{J}}^L}$ to sample $\mathsf{prer}^L_{i,j}$ for $(i,j) \in [K] \times [N^L] \setminus \hat{\mathcal{J}}^L$ as in $\mathbf{G}_{3,b}$. Continue the oracle simulation as in $\mathbf{G}_{3,b}$. According to this, if the simulation does not abort, send the key $k_{\hat{\mathbf{J}}^L}$ in the fourth message of the interaction.
- Let σ_b, σ_{1-b} be the local outputs of $\mathrm{O}_0, \mathrm{O}_1$, respectively. If $\sigma_0 = \quad \perp$ or $\sigma_1 = \quad \perp$, then run $b' \leftarrow \mathcal{A}(St', \perp, \perp)$. Else, run $b' \leftarrow \mathcal{A}(St', \sigma_0, \sigma_1)$ and output b'.

Note that if the values $\mathsf{prer}^L_{i,\hat{\mathbf{J}}^L_i}$ are random, then $\mathcal{B}$ perfectly simulates $\mathbf{G}_{4,b}$, whereas if they are the outputs of the pseudorandom function, $\mathcal{B}$ perfectly simulates $\mathbf{G}_{3,b}$. By the security of PRF with input length $\log(KN^L)$ we obtain

$$|\Pr[\mathbf{G}_{3,b} \Rightarrow 1] - \Pr[\mathbf{G}_{4,b} \Rightarrow 1]| \leq \epsilon_{\mathsf{PRF}}.$$

Game $\mathbf{G}_{5,b}$: In game $\mathbf{G}_{5,b}$, we change the way we generate the randomness seeds $\mathsf{prer}^R_{i,\hat{\mathbf{J}}^R_i}$ for $i \in [K]$. Concretely, we sample them at random from $\{0,1\}^n_{\mathsf{PRF}}$. Anologously to the previous change, a reduction from the security of PRF shows that

$$|\Pr[\mathbf{G}_{4,b} \Rightarrow 1] - \Pr[\mathbf{G}_{5,b} \Rightarrow 1]| \leq \epsilon_{\mathsf{PRF}}.$$

Game $\mathbf{G}_{6,b}$: In game $\mathbf{G}_{6,b}$, we change the way we compute the values $\mathsf{r}_{i,\hat{\mathbf{J}}_i}$ for $i \in [K]$. Note that in $\mathbf{G}_{5,b}$ these were computed as $\mathsf{r}_{i,\hat{\mathbf{J}}_i} := \mathsf{H}_x(\mathsf{prer}_{i,\hat{\mathbf{J}}_i})$. Now, we sample them randomly as

$$\mathsf{r}_{i,\hat{\mathbf{J}}_i} = (\alpha_{i,\hat{\mathbf{J}}_i}, \varphi_{i,\hat{\mathbf{J}}_i}, \gamma_{i,\hat{\mathbf{J}}_i}) \leftarrow\!\!\$\, \mathbb{Z}_p \times \mathcal{R}_{\mathsf{ck}} \times \{0,1\}^{n_{\mathsf{PRF}}}.$$

Note that $\mathcal{A}$ can only distinguish between games $\mathbf{G}_{5,b}$ and $\mathbf{G}_{6,b}$ if it queries $\mathsf{H}_x(\mathsf{prer}_{i,\hat{\mathbf{J}}_i})$ for some $i \in [K]$. However, $\mathcal{A}$ obtains no information about $\mathsf{prer}_{i,\hat{\mathbf{J}}_i}$, which is sampled uniformly at random. By a union bound over all hash queries, $i \in [K]$ and $\{L, R\}$ we obtain

$$|\Pr[\mathbf{G}_{5,b} \Rightarrow 1] - \Pr[\mathbf{G}_{6,b} \Rightarrow 1]| \leq \frac{2KQ_{\mathsf{H}_x}}{2^{n_{\mathsf{PRF}}}}.$$

Game $\mathbf{G}_{7,b}$: Game $\mathbf{G}_{7,b}$ is as $\mathbf{G}_{6,b}$, except that it computes the values $\mathsf{com}_{\mathsf{r},i}, i \in [K]$ in a different way. Concretely, for all $i \in [K]$, it samples $\eta_i \leftarrow\!\!\$\, \{0,1\}^n$ and computes the $\mathsf{com}_{\mathsf{r},i}$ as

$$\mathsf{com}_{\mathsf{r},i} := \mathsf{H}_{\mathsf{r}}(\mathsf{H}_{\mathsf{r}}(\mathsf{r}_{i,1}), \ldots, \mathsf{H}_{\mathsf{r}}(\mathsf{r}_{i,\hat{\mathbf{J}}_i-1}), \eta_i, \mathsf{H}_{\mathsf{r}}(\mathsf{r}_{i,\hat{\mathbf{J}}_i+1}), \ldots, \mathsf{H}_{\mathsf{r}}(\mathsf{r}_{i,N})).$$

Later it returns the η_i as part of its second message. Note that $\mathcal{A}$ can only see the difference between $\mathbf{G}_{6,b}$ and $\mathbf{G}_{7,b}$ if it queries $\mathsf{H}_\mathsf{r}(\mathsf{r}^X_{i,\hat{\mathbf{J}}_i^X})$ for an $i \in [K]$ and $X \in \{L, R\}$. It is clear that $\mathcal{A}$ obtains no information about $\gamma_{i,\hat{\mathbf{J}}_i}$ and $\gamma_{i,\hat{\mathbf{J}}_i}$ is sampled uniformly at random. Thus, a union bound over all $Q_{\mathsf{H}_\mathsf{r}}$ random oracle queries, $i \in [K]$, and $X \in \{L, R\}$ yields

$$|\Pr[\mathbf{G}_{6,b} \Rightarrow 1] - \Pr[\mathbf{G}_{7,b} \Rightarrow 1]| \leq \frac{2KQ_{\mathsf{H}_\mathsf{r}}}{2^{n_{\mathsf{PRF}}}}.$$

Game $\mathbf{G}_{8,b}$: In game $\mathbf{G}_{8,b}$ we change the way the commitments $\mu_{i,\hat{\mathbf{J}}_i}, i \in [K]$ are generated. Recall that before, these were generated as

$$\mu_{i,\hat{\mathbf{J}}_i} := \mathsf{Translate}(\mathsf{ck}, \mu_0, \varphi_{i,\hat{\mathbf{J}}_i}) = \mathsf{Com}(\mathsf{ck}, \mathsf{m}; \varphi_0 + \varphi_{i,\hat{\mathbf{J}}_i}).$$

Note that if the game does not stop, then especially $\hat{\mathbf{J}} = \mathbf{J}$ and $\varphi_i = \varphi_0 + \varphi_{i,\hat{\mathbf{J}}_i}$. In game $\mathbf{G}_{8,b}$, we sample $\varphi_i \leftarrow\!\!\$\, \mathcal{R}_{\mathsf{ck}}$ and set $\mu_{i,\hat{\mathbf{J}}_i} := \mathsf{Com}(\mathsf{ck}, \mathsf{m}; \varphi_i)$ for all $i \in [K]$. We claim that the view of $\mathcal{A}$ is unchanged. This is because, due to the previous changes, $\mathcal{A}$ gets no information about $\varphi_{i,\hat{\mathbf{J}}_i}$. Thus, we have to consider the distribution of the values $\varphi_i = \varphi_0 + \varphi_{i,\hat{\mathbf{J}}_i}$ conditioned on $k_{\hat{\mathbf{J}}}, (\varphi_0 + \varphi_{i,j})_{j \neq \hat{\mathbf{J}}_i}$ and φ_0. This distribution is uniformly random as $\varphi_{i,\hat{\mathbf{J}}_i}$ is uniformly random. Hence we have

$$\Pr[\mathbf{G}_{8,b} \Rightarrow 1] = \Pr[\mathbf{G}_{7,b} \Rightarrow 1].$$

Game $\mathbf{G}_{9,b}$: In game $\mathbf{G}_{9,b}$, we change the way μ_0 is generated, using that CMT is perfectly hiding. Concretely, we sample a random message $\bar{\mathsf{m}}^L$ (resp. $\bar{\mathsf{m}}^R$) and set $\mu_0 := \mathsf{Com}(\mathsf{ck}, \bar{\mathsf{m}}; \varphi_0)$. Note that in $\mathbf{G}_{9,b}$ the value φ_0 is only needed to compute μ_0. Especially, it is not needed to compute the values φ_i which are part of the final signatures due to the previous changes. It follows from the security of CMT that $\mathsf{Com}(\mathsf{ck}, \bar{\mathsf{m}}; \varphi_0)$ and $\mathsf{Com}(\mathsf{ck}, \mathsf{m}; \varphi_0)$ are identically distributed given ck. Therefore, the view of $\mathcal{A}$ is not changed and we get

$$\Pr[\mathbf{G}_{9,b} \Rightarrow 1] = \Pr[\mathbf{G}_{8,b} \Rightarrow 1].$$

Let us take a closer look at game $\mathbf{G}_{9,b}$. Here, for each instance $i \in [K]$, the only part that depends on message m (and hence bit b) is the $\mathbf{J}_i^{th}$ session. All other sessions only depend on μ_0, which does not depend on m anymore. Now, our final claim is that we can bound the difference between $\mathbf{G}_{9,0}$ and $\mathbf{G}_{9,1}$ using the perfect blindness under maliciously generated keys of the well-known BLS blind signatures scheme [4]. Concretely, we can now apply a straight-forward reduction from the game in Lemma 7 and obtain

$$\Pr[\mathbf{G}_{9,0} \Rightarrow 1] = \Pr[\mathbf{G}_{9,1} \Rightarrow 1].$$

In summary, we showed that $\mathbf{G}_{0,0}$ is close to $\mathbf{G}_{9,0}$, $\mathbf{G}_{9,0}$ is close to $\mathbf{G}_{9,1}$, and $\mathbf{G}_{9,1}$ is close to $\mathbf{G}_{0,1}$. Thus, $\mathbf{G}_{0,0}$ is close to $\mathbf{G}_{0,1}$, which is what we had to show. □

J Concrete Parameters of the Original Boosting Transform

Here we explain how we estimated the concrete efficiency for the boosting transform [30]. Concretely, we consider the Okamoto-Schnorr instantiation [31] of it. For simplicity, we ignore statistically negligible terms in the security loss. Also, we only focus on one-more unforgeability and not on blindness. In comparison with our schemes, this clearly favors [30].

With this in mind, we combine the concrete bounds given in [30] and obtain the following. For each adversary against the one-more unforgeability of the scheme that runs in time t, initiates at most q signature interactions, makes at most Q_{H} hash queries, and has success probability ϵ, there are algorithms solving the discrete logarithm problem in time $2t, t$ with success probability $\epsilon_{\mathsf{DLOG}}, \epsilon'_{\mathsf{DLOG}}$ such that

$$\epsilon \leq 4\left(\sqrt[3]{\epsilon_{\mathsf{DLOG}} Q_{\mathsf{H}}^2 \ell^3} + \frac{q}{2}\epsilon'_{\mathsf{DLOG}} + \frac{q^{\ell+1}}{|\mathbb{Z}_p|}\right),$$

where p is the order of the group and $\ell = 3\ln(q+1) + \ln(2/\epsilon)$. Assuming κ bits of security for the underlying discrete logarithm problem, this becomes

$$\epsilon \leq 4\left(\sqrt[3]{2t2^{-\kappa} Q_{\mathsf{H}}^2 \ell^3} + \frac{q}{2}t2^{-\kappa} + \frac{q^{\ell+1}}{2^{\kappa}}\right).$$

To compute a sufficiently large level κ, we consider every combination of ϵ and t such that $\epsilon/t = 2^{128}$ and find the minimum κ such that the above inequality leads to a contradiction. Then, we take the maximal of these.

We implemented the approach in a Python script, see Supplementary Material Section K.3.

K Scripts for Parameter Computation

Here, we present three Python scripts computing paramters of our schemes and a scheme obtained from the boosting transform. The scripts follow the high level approaches outlined in chapters H.5 and J and Section 4.4.

K.1 Parameter Script for Our RSA-based Scheme

Listing 1.1. Python Script to compute the parameters for our RSA-based scheme. A discussion can be found in Supplementary Material Section H.5.

```
#!/usr/bin/env python

import math
from tabulate import tabulate

#########################################################
# Functions to determine the RSA modulus length for a #
# given hardness, Formulas are taken from             #
# eprint.iacr.org/2019/260, Section 8.1               #
#########################################################
def heuristic_nfs_complexity(n, c, a):
        exponent = a*(math.log(n)**c)*((math.log(math.log(n)))**(1.0-c))
```

```
        return math.exp(exponent)

def tau(kappa):
        t = 1
        while 2**kappa > heuristic_nfs_complexity(2**(2*kappa*t), 1.0/3.0, (64/9.0)**(1.0/3.0)):
                t = t+1
        return t

def security_level_to_RSA_modulus_length(level):
        return 2*level*tau(level)




############################################################
# Functions to compute the bit sizes of signatures,        #
# keys and communication for given modulus, scalar space, #
#  commitment modulus and statistical security parameters #
############################################################

def size_pk(main_modulus_length, commitment_modulus_length, plambda_length):
        #size of parameters: main_modulus,
        #invertible element modulo main_modulus, plambda
        par_size = main_modulus_length + main_modulus_length + plambda_length

        #size of pk': range element, where range is Z_N^x
        #and N is the main_modulus
        pk_prime_size = main_modulus_length

        #size of commitment key: commitment_modulus,
        #element modulo commitment modulus, prime e
        # we assume e = 2^16+1
        ck_size = commitment_modulus_length + commitment_modulus_length + 17

        return par_size + pk_prime_size + ck_size

def size_sig(main_modulus_length, commitment_modulus_length, plambda_length):
        #signature consists of scalar, domain element and commitment randomness
        return plambda_length + (plambda_length + main_modulus_length) + commitment_modulus_length

#this returns coefficient of log(N) in the part of
#the communication that grows with log(N).
def size_communication_growing(main_modulus_length, commitment_modulus_length, plambda_length, secpar,
      ↪ secpar_prf):
        return 2 + secpar_prf

#this returns the part of
#the communication that does not grow with log(N).
def size_communication_constant(main_modulus_length, commitment_modulus_length, plambda_length, secpar,
      ↪ secpar_prf):
        return 4*secpar + plambda_length + main_modulus_length + commitment_modulus_length



################################################
# Main part of the script, computes level of RSA #
# needed to satisfy a given security level for   #
# the scheme for a given number of signatures    #
################################################


# Notation:
# epsilon : Success probability of adversary
# t : running time of adversary
# p : number of initiated interactions with signer oracle
# q_hash, q_hash_r, ... : number of queries for the respective hash function
# plambda_length : minumum bitlength of the prime lambda
#                  defining the scalar space of the underlying linear function
# level_main_rsa : security level of the main RSA instance
# level_commitment_rsa: security level of the RSA instance used for the commitment scheme



# Compute the right-hand side of the inequality upper bounding the success probability
# for an adversary against the omuf security of the scheme
def success_probability_upper_bound_omuf(log_epsilon, log_t, secpar, log_p, log_q_hash, log_q_hash_r,
      ↪ log_q_hash_c, log_q_hash_prime, log_q_hash_prime_prime ,plambda_length, level_main_rsa,
      ↪ level_commitment_rsa):

        p = 2**log_p
        q_hash = 2**log_q_hash
        q_hash_r = 2**log_q_hash_r
        q_hash_c = 2**log_q_hash_c
        q_hash_prime = 2**log_q_hash_prime

        #statistical terms in the reductions from BS to CCBS and from CCBS to EBS
        stat_term_1 = 2**(4*log_p-secpar) + 2**(3*log_p-secpar) + 2**(4*log_p-secpar) + 2**(3*log_p-secpar)
```

```
        stat_term_2 = 2**(2*log_q_hash_r-secpar) + 2**(2*log_q_hash_c-secpar) + 2**(log_p+log_q_hash_r-
              ↪ secpar) + 2**(log_p+log_q_hash_c-secpar) + 2**(log_p+log_q_hash_prime-secpar) + 2**(log_p+
              ↪ log_q_hash_prime_prime-secpar)

        #ell_BS: upper bound on the number of finished signature interactions of the linear BS scheme
        ell_BS = 3*math.log(p+1) + math.log(2) - math.log(2**log_epsilon-stat_term_2)
        log_ell_BS = math.log(ell_BS,2)

        log_term_a = 1+(2*log_q_hash+3*log_ell_BS+1+log_t-level_main_rsa)/3.0
        log_term_b = 1+(1+ell_BS)*(log_p+log_q_hash)-plambda_length
        log_term_c = 1+log_t-level_commitment_rsa
        log_term_d = 1+log_p+log_t-level_main_rsa

        total = 2**log_term_a + 2**log_term_b + 2**log_term_c + 2**log_term_d + 2*stat_term_1 + stat_term_2
        return total


# Compute an RSA level large enough such that
# epsilon <= success_probabilty_upper_bound_omuf ... leads to contradiction.
def rsa_level_from_epsilon_t_combination(level, log_epsilon, secpar, log_p, plambda_length):
        log_t = level + log_epsilon
        epsilon = 2**log_epsilon

        rhs = epsilon
        level_main_rsa = level
        level_commitment_rsa = level+10
        while rhs >= epsilon:
                level_main_rsa = level_main_rsa + 1
                #for simplicity, we set all hash query parameters to be the running time
                rhs = success_probability_upper_bound_omuf(log_epsilon, log_t, secpar, log_p, log_t, log_t,
                      ↪ log_t, log_t, log_t, plambda_length, level_main_rsa, level_commitment_rsa)

        return level_main_rsa



# Compute an RSA level large enough s.t. level bits of security are provided for omuf
def rsa_level_from_security_level(level, secpar, log_p, plambda_length):
        level_main_rsa = level

        # we consider every possible combination of epsilon and t and use the highest rsa level.
        for minus_log_epsilon in range(level+1):
                log_epsilon = -minus_log_epsilon
                l = rsa_level_from_epsilon_t_combination(level, log_epsilon, secpar, log_p, plambda_length)
                if l > level_main_rsa:
                        level_main_rsa = l

        return level_main_rsa


# Compute a secpar for prf large enough such that the blindness security bound leads to a contradiction.
def secpar_prf_from_epsilon_t_combination(level,log_epsilon,main_modulus_length,commitment_modulus_length,
      ↪ log_N_LR,secpar):
        log_t = level + log_epsilon
        epsilon = 2**log_epsilon

        rhs = epsilon
        secpar_prf = level
        while rhs >= epsilon:
                secpar_prf = secpar_prf + 1
                #for simplicity, we set all hash query parameters to be the running time
                rhs_term_1 = (2*log_N_LR-1)* 2**(log_t-secpar_prf+2)
                rhs_term_2 = 2**(2*log_t-secpar+1)
                rhs_term_3 = 2**(log_t-secpar+2)
                rhs_term_4 = 2**(log_t-secpar_prf+2)
                rhs_term_5 = 2**(log_t-secpar_prf+2)
                rhs_term_6 = 2**(log_t-secpar+2)
                rhs = rhs_term_1 + rhs_term_2 + rhs_term_3 + rhs_term_4 + rhs_term_5 + rhs_term_6

        return secpar_prf

# Compute a secpar for prf large enough s.t. level bits of security are provided for blindness
def secpar_prf_from_security_level(level,main_modulus_length,commitment_modulus_length,log_N_LR,secpar):

        secpar_prf = level

        # we consider every possible combination of epsilon and t and use the highest secpar_prf.
        for minus_log_epsilon in range(level+1):
                log_epsilon = -minus_log_epsilon
                l = secpar_prf_from_epsilon_t_combination(level,log_epsilon,main_modulus_length,
                      ↪ commitment_modulus_length,log_N_LR,secpar)
                if l > secpar_prf:
                        secpar_prf = l

        return secpar_prf
```

```
# returns one row of the final table
def table_row(level,log_p,plambda_length):
        secpar = 3*level
        # compute the level of RSA we need for omuf
        level_main_rsa = rsa_level_from_security_level(level,secpar,log_p,plambda_length)
        level_commitment_rsa = level+10

        # compute the modulus lengths for this level
        main_modulus_length = security_level_to_RSA_modulus_length(level_main_rsa)
        commitment_modulus_length = security_level_to_RSA_modulus_length(level_commitment_rsa)

        # compute the PRF security parameter we need for blindness
        # for simplicity, we upper bound N^L and N^R by the number of interactions p
        secpar_prf = secpar_prf_from_security_level(level,main_modulus_length, commitment_modulus_length,
              ↪ log_p,secpar)

        # compute key sizes, signature sizes and communication complexity
        pk = size_pk(main_modulus_length, commitment_modulus_length, plambda_length)
        sigma = size_sig(main_modulus_length, commitment_modulus_length, plambda_length)
        comm_grow = size_communication_growing(main_modulus_length, commitment_modulus_length,
              ↪ plambda_length, secpar, secpar_prf)
        comm_const = size_communication_constant(main_modulus_length, commitment_modulus_length,
              ↪ plambda_length, secpar, secpar_prf)

        # add this set of parameters to the table
        row = [level,log_p,secpar,secpar_prf,plambda_length,level_main_rsa,level_commitment_rsa,pk/8000.0,
              ↪ sigma/8000.0,comm_grow/8000.0,comm_const/8000.0]
        return row




#tabulate preparation
data = [["Level", "log p", "n", "n_PRF", "|lambda|", "Level RSA (main)", "Level RSA (com)", "|pk|", "|sigma|
      ↪ ", "Comm. a", "Comm. b"]]



#HERE you can insert the combinations you want to try.
levels = [80,128]
log_ps_class_a = [9]
plambda_lengths_class_a = [5000]
log_ps_class_b = [20]
plambda_lengths_class_b = [8000]
log_ps_class_c = [30]
plambda_lengths_class_c = [11000]


for level in levels:
        for log_p in log_ps_class_a:
                for plambda_length in plambda_lengths_class_a:
                        row = table_row(level,log_p,plambda_length)
                        data.append(row)

        for log_p in log_ps_class_b:
                for plambda_length in plambda_lengths_class_b:
                        row = table_row(level,log_p,plambda_length)
                        data.append(row)

        for log_p in log_ps_class_c:
                for plambda_length in plambda_lengths_class_c:
                        row = table_row(level,log_p,plambda_length)
                        data.append(row)


print(tabulate(data,headers='firstrow',tablefmt='fancy_grid'))
```

K.2 Parameter Script for Our CDH-based Scheme

Listing 1.2. Python Script to compute the parameters for our CDH-based scheme. A discussion can be found in Section 4.4.

```
#!/usr/bin/env python

import math
from tabulate import tabulate


######################################################################
# Functions to determine the (log of) group size for given hardness  #
```

```
# Formulas are taken from eprint.iacr.org/2019/260, Section 8.1     #
#########################################################################


def security_level_to_group_size_length(level):
        return 2*level+1

#########################################################################
# Functions to compute the bit sizes of signatures and keys and       #
# communication complexity for given group size, repetition parameter #
# K, commitment group size and statistical security parameters        #
#########################################################################

def size_pk(K, main_group_size_length, commitment_group_size_length):
        #group generator, K public keys (group elements), 2 group elements for the commitment
        return (K+1)*main_group_size_length + 2* commitment_group_size_length

def size_sig(K, main_group_size_length, commitment_group_size_length):
        #signature contains one aggregated group element and K times a commitment randomness
        return main_group_size_length + K * commitment_group_size_length

#this returns coefficient of log(N) in the part of the communication that grows with log(N).
def size_communication_growing(K, main_group_size_length, commitment_group_size_length, secpar, secpar_prf):
        return (1+K*secpar_prf)

#this returns the part of the communication that does not grow with log(N).
def size_communication_constant(K, main_group_size_length, commitment_group_size_length, secpar, secpar_prf)
      ↪ :
        return (K+5)*secpar + (K+1)*main_group_size_length + commitment_group_size_length + (K*math.log(K,2)
              ↪ +1-K)*secpar_prf



#########################################################################
# Main part of the script, computes level of security for DLOG needed  #
# to satisfy a given security level for the scheme for a given number  #
# of signatures                                                         #
#########################################################################


# Notation:
# epsilon : Success probability of adversary
# t : running time of adversary
# q : number of initiated interactions with signer oracle
# q_hash, q_hash_r, ... : number of queries for the respective hash function
# level_main_dlog : security level of the main DLOG/CDH instance
# level_commitment_dlog: security level of the DLOG/CDH instance used for the commitment scheme



# Compute the right-hand side of the inequality upper bounding the success probability
# for an adversary against the omuf security of the scheme
def success_probability_upper_bound(log_t, secpar, K, log_q, log_q_hash, log_q_hash_r, log_q_hash_c,
      ↪ log_q_hash_prime, level_main_dlog, level_commitment_dlog):

        #statistical term
        stat_term_a = 2**(2*log_q_hash_r-secpar)
        stat_term_b = 2**(2*log_q_hash_c-secpar)
        stat_term_c = 2**(log_q+log_q_hash_r-secpar)
        stat_term_d = K* 2**(log_q+log_q_hash_r-secpar)
        stat_term_e = 2**(log_q+log_q_hash_c-secpar)
        stat_term_f = 2**(log_q+log_q_hash_prime-secpar+1)
        stat_term = stat_term_a + stat_term_b + stat_term_c + stat_term_d + stat_term_e + stat_term_f

        term_a = 2**(-level_commitment_dlog+log_t)
        term_b = K*2**(-(2*level_main_dlog+1))
        term_c = 4*K*2**(log_q-level_main_dlog+log_t)
        term_d = stat_term

        total = 2*(term_a+term_b+term_c+term_d)
        return total


# Compute an dlog level large enough such that
# epsilon <= success_probability_upper_bound_ ... leads to contradiction.
def dlog_level_from_epsilon_t_combination(level, log_epsilon, secpar, log_q, K):
        log_t = level + log_epsilon
        epsilon = 2**log_epsilon

        rhs = epsilon
        level_main_dlog = level+10
        level_commitment_dlog = level+10
        while rhs >= epsilon:
                level_main_dlog = level_main_dlog + 1
                #for simplicity, we set all hash query parameters to be the running time
                rhs =  success_probability_upper_bound(log_t, secpar, K, log_q, log_t, log_t, log_t, log_t,
                      ↪ level_main_dlog, level_commitment_dlog)

        return level_main_dlog
```

```
# Compute an DLOG level large enough s.t. level bits of security are provided
def dlog_level_from_security_level(level, secpar, log_q, K):
        level_main_dlog = level

        # we consider every possible combination of epsilon and t and use the highest dlog level.
        for minus_log_epsilon in range(level+1):
                log_epsilon = -minus_log_epsilon
                l = dlog_level_from_epsilon_t_combination(level, log_epsilon, secpar, log_q, K)
                if l > level_main_dlog:
                        level_main_dlog = l

        return level_main_dlog




# Compute a secpar for prf large enough such that the
# blindness security bound leads to a contradiction.
def secpar_prf_from_epsilon_t_combination(level,log_epsilon,log_N_LR,K,secpar):
        log_t = level + log_epsilon
        epsilon = 2**log_epsilon

        rhs = epsilon
        secpar_prf = level
        while rhs*2**(log_t) >= epsilon:
                secpar_prf = secpar_prf + 1
                #for simplicity, we set all hash query parameters to be the running time
                rhs_term_1 = (2*log_N_LR + 2*math.log(K,2)-1)*K*2**(log_t-secpar_prf+2)
                rhs_term_2 = 2**(2*log_t-secpar+1)
                rhs_term_3 = 2**(log_t-secpar+2)
                rhs_term_4 = K* 2**(log_t-secpar_prf+2)
                rhs_term_5 = K* 2**(log_t-secpar_prf+2)
                rhs = rhs_term_1 + rhs_term_2 + rhs_term_3 + rhs_term_4 + rhs_term_5

        return secpar_prf

# Compute a secpar for prf large enough s.t. level bits of security are provided for blindness
def secpar_prf_from_security_level(level,log_N_LR,K,secpar):

        secpar_prf = level

        # we consider every possible combination of epsilon and t and use the highest secpar_prf.
        for minus_log_epsilon in range(level+1):
                log_epsilon = -minus_log_epsilon
                l = secpar_prf_from_epsilon_t_combination(level,log_epsilon,log_N_LR,K,secpar)
                if l > secpar_prf:
                        secpar_prf = l

        return secpar_prf




# checks the condition that vartheta and K have to satisfy in order to apply the OMUF theorem
# for the security level we aim to achieve
def vartheta_from_constraint(level,K):
        denom = 1.0 - (math.log(2**(level+1))/float(K))
        return (1.0/denom) + 0.1


#returns the minimum integer K such that there even exists a positive vartheta
def minimum_plausible_K(level):
        return int(math.log(2**(level+1))+1)
        return int(math.log(2**(level+1))/2.0+1)

# returns one row of the final table
def table_row(level,log_q,K):
        secpar = 4*level

        # compute the vartheta we need to satisfy the constraint
        vartheta = vartheta_from_constraint(level,K)
        if vartheta <= 0:
                return []

        # compute the level of DLOG we need
        level_main_dlog = dlog_level_from_security_level(level,secpar,log_q,K)
        level_commitment_dlog = level+10

        # compute the group elements lengths for this level
        main_group_size_length = security_level_to_group_size_length(level_main_dlog)
        commitment_group_size_length = security_level_to_group_size_length(level_commitment_dlog)

        # compute the PRF security parameter we need for blindness
        # for simplicity, we upper bound N^L and N^R by the number of interactions q
        secpar_prf = secpar_prf_from_security_level(level,log_q,K,secpar)
```

```
        # compute key sizes, signature sizes and communication complexity
        pk = size_pk(K, main_group_size_length, commitment_group_size_length)
        sigma = size_sig(K, main_group_size_length, commitment_group_size_length)
        comm_grow = size_communication_growing(K, main_group_size_length, commitment_group_size_length,
              ↪ secpar, secpar_prf)
        comm_const = size_communication_constant(K, main_group_size_length, commitment_group_size_length,
              ↪ secpar, secpar_prf)

        # add this set of parameters to the table
        row = [level,log_q,secpar,secpar_prf,vartheta,K,level_main_dlog,level_commitment_dlog,pk/8000.0,
              ↪ sigma/8000.0,comm_grow/8000.0,comm_const/8000.0]
        return row




#HERE you can insert the combinations you want to try.
levels = [80,128]
log_qs = [20,30]

#tabulate preparation
data = [["Level", "log_q", "n", "n_PRF", "vartheta", "K", "Level DLOG (main)", "Level DLOG (com)", "|pk|", "
      ↪ |sigma|", "Comm. a", "Comm. b"]]
print("")

for level in levels:
        for log_q in log_qs:
                K_init = minimum_plausible_K(level)
                for K_off in range(0,30,10):
                        K = K_init+K_off
                        row = table_row(level,log_q,K)
                        data.append(row)

print(tabulate(data,headers='firstrow',tablefmt='fancy_grid'))
```

K.3 Parameter Script for the Boosting Transform

Listing 1.3. Python Script to compute the parameters for the Okamoto-Schnorr instantiation of the boosting transform. A discussion can be found in Supplementary Material Section J.

```
#!/usr/bin/env python

import math
from tabulate import tabulate


######################################################################
# Functions to determine the (log of) group size for given hardness  #
# Formulas are taken from eprint.iacr.org/2019/260, Section 8.1      #
######################################################################


def security_level_to_group_size_length(level):
        return 2*level+1

######################################################################
# Functions to compute the bit sizes of signatures and keys and      #
# communication complexity for given group size, repetition parameter #
# K, commitment group size and statistical security parameters       #
######################################################################

def size_pk(group_size_length):
        #group generator,  public key (group element)
        return 2*group_size_length

def size_sig_schnorr(group_size_length, commitment_randomness_length):
        #signature contains c',s', and a commitment randomness
        return 2*group_size_length + commitment_randomness_length

def size_sig_okamoto_schnorr(group_size_length, commitment_randomness_length):
        #signature contains c',s_1',s_2', and a commitment randomness
        return 3*group_size_length + commitment_randomness_length


######################################################################
# Main part of the script, computes level of security for DLOG needed  #
# to satisfy a given security level for the scheme for a given number  #
# of signatures                                                      #
######################################################################
```

```
# Notation:
# epsilon : Success probability of adversary
# t : running time of adversary
# q : number of initiated interactions with signer oracle
# level_dlog : security level of the underlying DLOG instance

# Compute the right-hand side of the inequality upper bounding the success probability
# for an adversary against the omuf security of the scheme
def success_probability_upper_bound_omuf(log_epsilon, log_t, log_q, level_dlog):
        q = 2**log_q

        #ell_BS: upper bound on the number of finished signature interactions of the linear BS scheme
        ell_BS = 3*math.log(q+1) + math.log(2) - math.log(2**log_epsilon)

        term1 = ell_BS * 2**(2+(1+log_t-level_dlog+2*log_t)/3.0)
        term2 = 2**(log_q + 1 + log_t - level_dlog)
        term3 = 2**(log_q*(ell_BS+1) -level_dlog)

        return term1 + term2 + term3



# Compute a DLOG level large enough such that
# epsilon <= success_probabilty_upper_bound_omuf ... leads to contradiction.
def dlog_level_from_epsilon_t_combination(level, log_epsilon, log_q):
        log_t = level + log_epsilon
        epsilon = 2**log_epsilon

        rhs = epsilon
        # if we started from level, we would result in overflows as the RHS is too large.
        level_dlog = 47*level
        while rhs >= epsilon:
                level_dlog = level_dlog + 1
                rhs = success_probability_upper_bound_omuf(log_epsilon, log_t, log_q, level_dlog)

        return level_dlog



# Compute a DLOG level large enough s.t. level bits of security are provided for omuf
def dlog_level_from_security_level(level, log_q):
        level_dlog = level

        # we consider every possible combination of epsilon and t and use the highest rsa level.
        for minus_log_epsilon in range(level+1):
                log_epsilon = -minus_log_epsilon
                l = dlog_level_from_epsilon_t_combination(level, log_epsilon, log_q)
                if l > level_dlog:
                        level_dlog = l

        return level_dlog




level = 128
log_q = 30

commitment_randomness_length = 128
level_dlog = dlog_level_from_security_level(level,log_q)
group_size_length = security_level_to_group_size_length(level_dlog)
size_pk = size_pk(group_size_length)
size_sig_schnorr = size_sig_schnorr(group_size_length, commitment_randomness_length)
size_sig_okamoto_schnorr = size_sig_okamoto_schnorr(group_size_length, commitment_randomness_length)

print("Want to support q = 2^" + str(log_q) + " signatures.")
print("==> Need level for DLOG >= " + str(level_dlog))
print("==> Need group bit size for DLOG >= " + str(group_size_length))
print("==> Public Key Size (in KB) >= " + str(size_pk/8000.0))
print("==> Schnorr Signature Size (in KB) >= " + str(size_sig_schnorr/8000.0))
print("==> Okamoto-Schnorr Signature Size (in KB) >= " + str(size_sig_okamoto_schnorr/8000.0))
```