PI-Cut-Choo! Parallel Instance Cut and Choose for Practical Blind Signatures

Benedikt Wagner, Lucjan Hanzlik, and Julian Loss

CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
{benedikt.wagner,hanzlik,loss}@cispa.de

Abstract. Known constructions of (efficient) blind signatures either rely on non-standard hardness assumptions or require parameters that grow linearly with the number of concurrently issued signatures. This holds true even in the random oracle model.
Katz, Loss and Rosenberg (ASIACRYPT 2021) presented a generic construction that boosts a scheme supporting logarithmically many concurrent signing sessions to a scheme that supports polynomially many. Unfortunately, this construction has two drawbacks: 1) the communication between the signer and the user still grows linearly with the number of issued signatures 2) their schemes inherit a very loose security bound from the underlying scheme and, as a result, require impractical parameter sizes.
In this paper, we eliminate these two drawbacks by proposing two highly practical blind signature schemes from the CDH and RSA assumptions. Our resulting schemes have communication which grows only logarithmically in the number of issued signatures. In addition, we introduce new techniques to mitigate the large security loss in the construction of Katz et al. Overall, we obtain the following parameter sizes (providing 128 bits of security):

- Our main scheme Pika is based on the BLS blind signature scheme (Boldyreva, PKC 2003) and is secure under the CDH assumption over a standard-sized group. Signatures are of size roughly 3 KB and communication per signature is roughly 150 KB.
- Our RSA-based scheme is based on the Okamoto-Guillou-Quisquater blind signature scheme (Okamoto, CRYPTO 1992). It has signatures and communication of roughly 9 KB and 8 KB, respectively.

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

1 Introduction

In 1982, David Chaum introduced blind signature schemes in the context of electronic cash [9]. 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 [24,34]. *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 [9,30], anonymous credentials [6,7] and e-voting [19]. Recently, blind signatures have also been used to add privacy features to blockchain-based systems [23]. 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 [4,2,12] or have parameters that grow linearly in the number of concurrent signing sessions [34,21,3,25]. 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. Blind signature schemes can be built generically from any secure signature scheme using secure two-party computation [24] or commitment schemes and zero-knowledge proofs [11]. The drawback of the construction in [24] is that the scheme is that signatures must be issued *sequentially.* However, typically one aims for the stronger notion of concurrent security. While [11] indeed achieves the latter, neither of these generic constructions are known to be efficiently instantiable. 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. As already discussed, these schemes are either inefficient as they rely on complexity leveraging [15] or rely on strong q-type or non-interactive assumptions [29,16,12,17].

Unfortunately, even in the random oracle model, the situation does not improve much. While there are simple constructions [4,2,34,21,22], they either require similar assumptions as their standard model counterparts [4,2] or support only a very small number of signatures per public key [34,21,22,3].

As a first step to overcome these limitations, Katz et al. [25] 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 [32]. 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.

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 Our Contribution

We answer this question in the affirmative by providing two new blind signatures schemes from the RSA and CDH assumptions. Both of our schemes follow a common high-level template which we will outline in the next section. In a nutshell, we start with the boosting transform [25] and eliminate its main drawback, namely the linearly growing communication complexity. Instead, the communication size of our schemes only grows logarithmically with the number of interactions. It should be emphasized that this dependency is very weak, as the coefficient of the logarithmic term only depends on statistical security parameters and not on the computational hardness assumptions. For each of these schemes, we give a concrete security analysis and show how to efficiently achieve a security level of 128 bit. We summarize our results below.

- Our scheme PIKA from CDH is based on the BLS blind signature scheme [5,4]. PIKA supports 2^{40} signing interactions with signatures of size 3 KB and communication complexity roughly 150 KB. It can be run over a standard elliptic curve group, e.g. P-384 [27].
- Our scheme from RSA is based on the linear blind signature scheme from the OGQ function [28,33,21]. We present parameters that allow for 2^{32} signing interactions while having signatures of size 9 KB and communication complexity roughly 8 KB.

Example parameters of our schemes can be found in Table 1. For an explanation, see Sections 3.3 and 4.3 and Supplementary Material Section J.

Finally, we note that our high level template can easily be generalized to schemes that can be obtained from linear function families [21].

Scheme	Signatures	$\lvert\mathsf{pk}\rvert$	$\lvert\sigma\rvert$	a	b	Max
$\mathsf{BS}_{\mathsf{RSA}}$	2^{20}	18.37	7.91	0.02	7.11	7.79
$\mathsf{BS}_{\mathsf{RSA}}$	2^{32}	18.87	8.91	0.01	7.61	8.30
$\mathsf{BS}_{\mathsf{CDH}}$	2^{20}	3.68	3.16	3.04	26.50	148.45
$\mathsf{BS}_{\mathsf{CDH}}$	2^{40}	4.13	3.16	3.06	27.01	149.42

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.

1.2 Technical Overview

We now give a summary of our techniques. To this end, let us recall the main ideas of the boosting transform CCBS [25] as well as its drawbacks. We start

with a linear blind signature scheme BS. For the purpose of this overview, it is not important what exactly a linear blind signature scheme is, we only state its properties of relevance, which are as follows:

1. The scheme satisfies one-more unforgeability (OMUF), as long as only a logarithmic number of signatures are issued.
2. If a reduction knew the randomness ur and the message m that is input into the user algorithm U and controls the random oracle, it could simulate the signer algorithm without knowing the secret key (using HVZK and random oracle programming).
3. The signature issuing protocol consists of three messages R, c, s, called the commitment, challenge and response, respectively.

The Boosting Transform. Using these features, the high-level idea of the boosting transform is to make the user commit to its random coins using a random oracle. Later, the user has to open these commitments in cut-and-choose fashion.

In more detail, 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 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 blind signature scheme BS, where the user inputs μ_j, ur_j in the j-th session.
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 BS as well as the randomness φ_J which binds m to μ_J.

We recall the high-level idea used in the OMUF proof of the boosting transform. The idea is to prove a reduction from the security of CCBS to the security of BS. The main challenge here is that the OMUF game for the underlying scheme BS only allows for a logarithmic number of signing interactions. On the other hand, 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 for an $j \in [N]$, the reduction can extract these values from the commitments com_j by observing the random oracle queries. Then, using the property 2 of the underlying scheme, it can later provide the correct response s_j. On the other hand, if the adversary in CCBS *cheats*

(i.e., it malforms the commitment for the Jth session in the first step and is not caught) then the reduction may not be able to extract such values for the cut and choose index J. In this case, the reduction is not able to rely on programming in order to provide the correct answer s_J. Instead, it has to rely on the signing oracle of the underlying OMUF game for BS.

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

$$\sum_{N=1}^{p+1} \frac{1}{N} < \ln(p+1), \tag{1}$$

which is logarithmic. 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. This is the main insight of the boosting transform.

Although the boosting transform exponentially increases the security of the underlying blind signature scheme BS, this comes at a steep price in terms of efficiency: 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.

This dependency on N in the communication complexity arguably renders CCBS impractical. In addition, in the above reduction from the OMUF security of BS, the number of signing queries to the underlying signing oracle behaves as $\ln(1/\epsilon)$. Here, ϵ is the advantage of the adversary in the OMUF game of CCBS. If ϵ is small (say, 2^{-128}), one has to pick very impractical parameter sizes for BS.

Puncturable Pseudorandom Functions to the Rescue. Our first objective is to eliminate this linear dependency on N and improve it by an at most logarithmic dependency.

In our first step we focus on d). Here, our idea is to a leverage a puncturable pseudorandom function (PPRF)[1] to generate the randomness ur_j, φ_j. Namely, the user chooses a key k for a PPRF and generates $(\mathsf{ur}_j, \varphi_j) := \mathsf{Eval}(k, j)$. In this way, the user can open the commitments com_j just by sending $\mu_j, j \in [N]$ and k_J, where k_J is a key punctured at position J. Of course, the linear dependency still remains due to the values μ_j that have to be sent. Our solution here is to replace the unstructured commitment $\mu_j = \mathsf{H}(\mathsf{m}, \varphi_j)$ with a more structured commitment scheme Com, which can be (publicly) rerandomized. To be more precise, we make use of a commitment scheme[2] in which the randomness space

[1] This PPRF can be instantiated in a statistically secure way by applying the GGM construction [18] to a random oracle.

[2] We show that such commitments can easily be constructed tightly based on standard assumptions.

forms a group and, given only μ and φ', one can publicly transform $\mu := \mathsf{Com}(\mathsf{m}, \varphi)$ into $\mathsf{Com}(\mathsf{m}, \varphi + \varphi')$, . Now, the user first sends an initial commitment $\mu_0 := \mathsf{Com}(\mathsf{m}, \varphi_0)$. Using the properties of the commitment scheme, the signer can then efficiently compute $N-1$ commitments from μ_0 and the punctured key k_J via $\mu_j = \mathsf{Com}(\mathsf{m}, \varphi_0 + \varphi_j)$. Note that due to the property of the commitment scheme, the signer can compute the commitments μ_j for $j \neq J$ just from μ_0. Intuitively, this preserves blindness, as the punctured key k_J does not reveal anything about the randomness ur_J, φ_J.

Next, we eliminate the linear dependency a). Here, we replace the commitments $\mathsf{com}_j = \mathsf{H}(\mathsf{ur}_j, \mu_j)$ by a single Merkle tree commitment com_r, committing to all ur_j, μ_j at once. To be precise, we let the j-th leaf of the Merkle tree be a salted hash of ur_j, μ_j. In this way, the user can later open the commitment by providing the punctured key k_J and the salted hash of ur_J, μ_J. We can use the same trick to address the dependency in c).

Finally, we tackle b). Our first idea is to make the signer send a random seed seed_R to the user. Then both can compute the commitments R_j via $R_j = \mathsf{H}(\mathsf{seed}_R, j)$. In the boosting transform, the values R_j were computed as $R_j = \mathsf{F}(r_j)$, where F is a linear one-way hash function. The value r_J is needed to compute the final response s_J. Unfortunately, the signer can not know r_J if we define R_J to be the output of a random oracle. In the case of our RSA-based scheme we can easily overcome this problem by giving the signer an appropriate trapdoor for the function F as part of its secret key.

A Scheme from CDH. The rest of this overview focuses on our CDH-based scheme Pika. Here, we start from the BLS blind signature scheme [4]. Although this scheme does not fall into the class of linear blind signature schemes, we observe that it has property 2. Interestingly, the scheme only has two rounds and so linear dependency b) disappears.

We emphasize that the original BLS blind signature scheme is secure under a one-more variant of the CDH assumption. Hence, we can not directly apply our technique to this construction if we are aiming for a scheme based on the (plain) CDH assumption.

To overcome this issue, we now introduce our second key idea. Recall that in the security proof of the original boosting transform, one has to access the signer oracle of the underlying scheme a logarithmical amount of times, due to Equation (1). We observe that by letting the cut-and-choose parameter grow slightly faster than before, the expected number of successful cheats can be bounded by a constant. We can even force the expectation to be less than 1 by scaling appropriately. However, his does not by itself resolve the issue that the actual number of cheats can still deviate as much as $\ln(1/\epsilon)$ from its expectation (where again ϵ denotes the advantage of the adversary in the OMUF game). We can, however, use the Chernoff bound to show that exceeding a single cheat happens with some constant probability less than 1. (We remark that this could also be shown using Markov's inequality, but would lead to a far looser bound.)

Then, we play our next card, which is parallel repetition. Namely, we let the signer and the user run K completely independent instances of our scheme so far,

where each instance is relative to a separate secret key. We show that with high probability, in one randomly chosen instance $i^* \in [K]$, there is *no cheat at all*. Using this observation, we can now give a reduction from the key-only security of the underlying blind signature scheme to finish our proof. We observe that for the BLS blind signature scheme, key-only security can be shown directly from CDH. In this fashion, we successfully avoided the use of a one-more assumption. The key benefit of this construction is that the BLS scheme allows for efficient aggregation of signatures (with distinct keys) on the same message. Hence, it is easy to merge the resulting signatures from the K instances for a significant efficiency improvement! An additional property of PIKA is that it can be used over a standard-sized group (e.g., P-384 [27]). This makes the scheme a much more practical choice than its Schnorr-based counterparts.

Further Optimizations. In our RSA-based scheme, we reduce directly from the security of the CCBS construction instantiated with the OGQ linear function [28]. Thus, without further modification, our scheme inherits the impractical parameter sizes incurred by the dependency on the term $\ln(1/\epsilon)$. Recall that the OGQ linear function maps from the domain $\mathbb{Z}_\lambda \times \mathbb{Z}_N^*$ to the range $\mathbb{Z}_N^*$. Here, N is an RSA modulus and λ is a large prime number. Our observation is that we can increase the size of λ independently of N so as to increase the number of signatures supported by the base scheme BS. Compared to a naive increase of parameter sizes, this leads to a very practical scheme. We introduce further minor optimizations that help to keep the size of the puncturable PRFs output (and hence communication) low.

Second, we note that the blindness proof in [25] has a quadratic security loss in the number of signing sessions. This is because the reduction has to guess the cut-and-choose index before reducing from the underlying scheme BS. For our RSA-based scheme, this is not much of a problem due to statistical blindness of the underlying scheme. However, when we apply the same proof strategy to our scheme PIKA with parallel repetition, this loss becomes exponential in K, the number of parallel repetitions. Making up for this large loss would require very large statistical security parameters and thus bad efficiency. Fortunately, we can improve this significantly by making the signer commit to its cut-and-choose indices in the first message of the interaction. In this way, the reduction can extract these indices by observing the random oracle rather than guessing them.

1.3 Concurrent Work

In independent and concurrent work, Chairattana-Apirom and Lysyanskaya [8] also provide an improved boosting construction. Using very similar ideas as ours, they also improve the communication complexity from linear to logarithmic. While their transform can be generically applied to any linear function family in the sense of [21], it inherits the loose security bounds and poor parameter sizes from [25]. To highlight the disparity, we computed the parameter sizes for their version of the Okamoto-Schnorr [28] scheme whose security is based on the discrete logarithm assumption. Our calculations show that in order to support

2^{40} signatures, their scheme requires a 7072 bit group and yields signatures of size roughly 5.3 KB. It is apparent that these sizes do not favorably compare with the sizes of our CDH-based scheme Pika. First, their signatures are roughly 78% larger than ours. Second, their scheme requires impractical group sizes for which no standardized elliptic curve groups exist. By comparison, Pika can be implemented over an optimized 384-bit group which leads to a computationally far more efficient scheme.

We also computed parameter sizes for a Schnorr-based version of their blind signature scheme. We remark that Katz et al. [25] also provide a parameter estimate at the 128 bit security level, but this holds only for the specific choice of 2^{17} signing queries, 2^{80} random oracle queries and advantages above 2^{-23}. For more general parameters and 2^{40} signatures, this instantiation still yields signature sizes of 3.5 KB and requires a 7072 bit group. In addition, the analysis requires the AGM [13,14] and requires the (interactive) One-More Discrete Logarithm assumption.

For a detailed explanation of our calculations, see Supplementary Material Section I.

2 Preliminaries

Notation. The security parameter is denoted by $n \in \mathbb{N}$ and 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 probability distribution $\mathcal{D}$, we write $x \leftarrow \mathcal{D}$ if x is sampled according to $\mathcal{D}$. Similarly, 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 $y \in \mathcal{A}(x)$ to indicate 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 experiment $\mathbf{G}$, 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 $\mathbb{Z}_N^*$.

Cryptographic Background. In this section we introduce puncturable pseudorandom functions, a class of commitment schemes and the main object of interest, namely blind signature schemes. We recall the necessary computational assumptions in Supplementary Material Section A.

For the definition of puncturable pseudorandom functions, we follow [36].

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* $r \in \mathcal{R} = \{0,1\}^n$.

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:*

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

We define a special type of perfectly hiding commitment scheme in which the randomness can be rerandomized publicly.

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* $\mathsf{ck} \in \mathsf{Gen}(1^n)$, *the following distributions are identical:*

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

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:*

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

Randomness homomorphism is not a strong requirement for commitment schemes. Indeed, such commitment schemes can easily be derived from linear identification schemes using a folklore transformation.

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.*
- $\mathsf{Ver}(\mathsf{pk}, \mathsf{m}, \sigma)$ *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*

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

Definition 4 ((Honest Signer) Blindness). *Let $\mathsf{BS} = (\mathsf{Gen}, \mathsf{S}, \mathsf{U}, \mathsf{Ver})$ be a blind signature scheme. 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 $\sigma_0 = \bot$ or $\sigma_1 = \bot$, then run $b' \leftarrow \mathcal{A}(St', \bot, \bot)$. Else, run $b' \leftarrow \mathcal{A}(St', \sigma_0, \sigma_1)$.*
4. *Output b'.*

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

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

Definition 5 (One-More Unforgeability). *Let $\mathsf{BS} = (\mathsf{Gen}, \mathsf{S}, \mathsf{U}, \mathsf{Ver})$ be a blind signature scheme. For an adversary $\mathcal{A}$, we consider the following game $\mathbf{OMUF}^{\mathcal{A}}_{\mathsf{BS}}(n)$:*

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 *arbitrary often and interleaved. Let* ℓ *denote the number of interactions that* $\mathcal{A}$ *completed with* O.
3. *Output* 1 *if and only if all* $\mathsf{m}_i, i \in [k]$ *are distinct,* $k > \ell$ *and for each* $i \in [k]$ *it holds that* $\mathsf{Ver}(\mathsf{pk}, \mathsf{m}_i, \sigma_i) = 1$.

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

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

We formally introduce Merkle Trees in Supplementary Material Section B. Informally, for a random oracle H, we denote the computation of the root of the Merkle Tree over values $x_1, \ldots, x_\ell$ using H by

$$\mathsf{com} := \mathsf{tree}^{\mathsf{H}}(x_1, \ldots, x_\ell).$$

If for some value x_I only the hash $h = \mathsf{H}(x_J)$ is given, we denote the computation of the root by

$$\mathsf{com} := \mathsf{ptree}^{\mathsf{H}}_I(x_1, \ldots, x_{I-1}, \mathsf{H}(x_I), x_{I+1}, \ldots, x_\ell).$$

3 Scheme based on BLS

Here, we construct a blind signature scheme based on the CDH assumption. The starting point is the BLS signature scheme [5] and its blind variant [4].

3.1 Construction

Let $\mathsf{PGGen}(1^n)$ be a bilinear group generation algorithm. We assume that $\mathsf{PGGen}(1^n)$ outputs a cyclic group $\mathbb{G}$ of prime order p with generator g. Also, $\mathsf{PGGen}(1^n)$ outputs a pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ into some target group $\mathbb{G}_T$. 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). We also need random oracles $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_p, \mathsf{H}' : \{0,1\}^* \to \{0,1\}^n$ and $\mathsf{H}_\mathsf{r}, \mathsf{H}_c : \{0,1\}^* \to \{0,1\}^n, \mathsf{H}_x : \{0,1\}^* \to \mathbb{Z}_p \times \mathcal{R}_{\mathsf{ck}} \times \{0,1\}^{n_{\mathsf{PRF}}}$.

Our CDH-based 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 $f : \mathbb{N} \to \mathbb{N}$. Roughly, the function f determines how fast the cut-and-choose parameter N grows. We carefully set these parameters in Section 3.3.

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

1. Generate parameters $(\mathbb{G}, g, p, e) \leftarrow \mathsf{PGGen}(1^n)$ as above.
2. For each instance $i \in [K]$, sample $\mathsf{sk}_i \leftarrow\!\!\$\, \mathbb{Z}_p$ and set $\mathsf{pk}_i := g^{\mathsf{sk}_i}$.
3. Sample a commitment key $\mathsf{ck} \leftarrow \mathsf{CMT}.\mathsf{Gen}(1^n)$.
4. Return the public key $\mathsf{pk} := (\mathbb{G}, g, p, e, \mathsf{pk}_1, \ldots, \mathsf{pk}_K, \mathsf{ck})$ and the 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 1 and 2. As in our RSA-based scheme, S keeps a state ctr, which is inititalized as $\mathsf{ctr} := 1$.

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

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

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

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

Fig. 1. The algorithm Check used in the issuing protocol of blind signature scheme $\mathsf{BS}_{\mathsf{CDH}}$, 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.

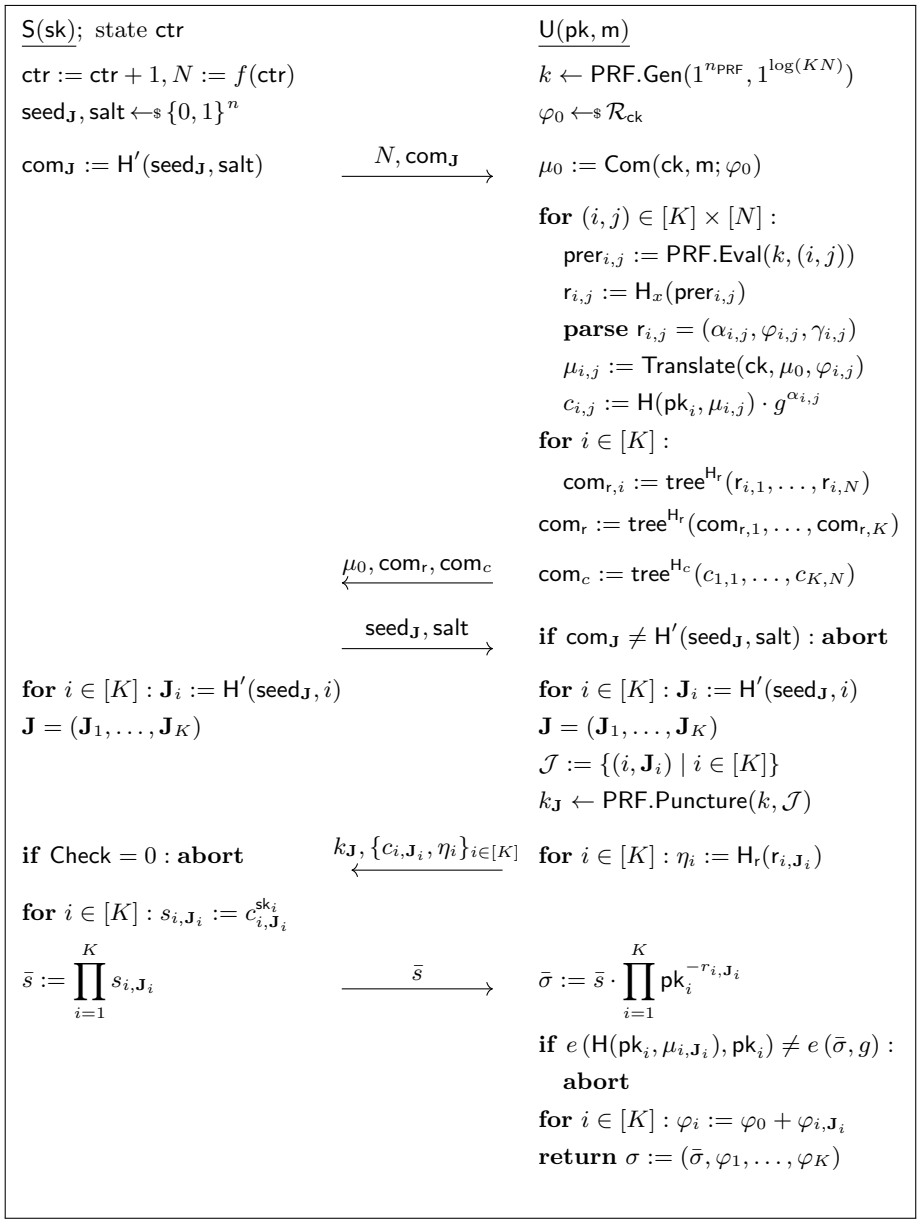

Fig. 2. The signature issuing protocol of the blind signature scheme $\mathsf{BS}_{\mathsf{CDH}}$, where $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_p, \mathsf{H}' : \{0,1\}^* \to \{0,1\}^n$ and $\mathsf{H}_\mathsf{r}, \mathsf{H}_c : \{0,1\}^* \to \{0,1\}^n, \mathsf{H}_x : \{0,1\}^* \to \mathbb{Z}_p \times \mathcal{R}_\mathsf{ck} \times \{0,1\}^{n_\mathsf{PRF}}$ are random oracles. The algorithm Check is defined in Figure 1. The state ctr of S is incremented atomically.

3.2 Security Analysis

Completeness of the scheme follows by inspection.

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

Theorem 2. *Let* CMT *be a randomness homomorphic commitment scheme and* PRF *be a puncturable pseudorandom function. Let* $\mathsf{PGGen}(1^n)$ *be a bilinear group 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 BS_CDH *satisfies one-more unforgeability, under the* CDH *assumption relative to* PGGen.

Specifically, assume the existence of an adversary against the OMUF *security of* BS_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

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

Proof. Set $\mathsf{BS} := \mathsf{BS}_\mathsf{CDH}$. 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 $\mathbf{G}_0 := \mathbf{OMUF}^{\mathcal{A}}_{\mathsf{BS}}$, 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. As we consider the random oracle model, $\mathcal{A}$ also gets access to random oracles $\mathsf{H}, \mathsf{H}', \mathsf{H}'', \mathsf{H}_\mathsf{r}$ 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 $\mathbf{G}_0$ and $\mathbf{G}_1$ if such a collision happens, i.e. H_t returns $\perp$. We can apply a union bound over all $Q^2_{\mathsf{H}_t}$ pairs of random oracle queries and obtain

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

Game $\mathbf{G}_3$: In game $\mathbf{G}_3$ we add another change to the random oracles $\mathsf{H}_t, t \in \{\mathsf{r}, c\}$. We again sample $y \leftarrow\!\!\$\, \{0,1\}^n$ if $\mathsf{H}_t(x)$ is not yet defined. This time, we also check if $\mathsf{H}_t(y)$ is already defined and return $\perp$ if this is the case. The detailed behavior of oracle H_t can be found in Figure 5. Note that the adversary $\mathcal{A}$ can only distinguish between $\mathbf{G}_1$ and $\mathbf{G}_2$ if such a chain happens, i.e. H_t returns $\perp$ because $\mathsf{H}_t(y)$ was already defined. Again, we can apply a union bound over all $Q^2_{\mathsf{H}_t}$ pairs of random oracle queries and obtain

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

To summarize the changes we did so far, we ruled out two patterns for random oracle queries:

1. Collisions: $\exists x \neq x' : \mathsf{H}_t(x) = \mathsf{H}_t(x')$.
2. Chains: $\exists x$: query $\mathsf{H}_t(\mathsf{H}_t(x))$ was made before query $\mathsf{H}_t(x)$.

In particular, this 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. Further, note that for any $x_0 \in \{0,1\}^n$ the sequence $(x_i)_i$ with $x_i := \mathsf{H}_t^{-1}(x_{i-1})$ for $i \in \mathbb{N}$ does not contain any repeating value. Indeed, such a cycle could only be produced if the adversary managed to form a chain, which was ruled out. Note that this implies that algorithm $\mathsf{ExtLeafs}_t$ given in Figure 5 always terminates. Roughly, this algorithm extracts leafs from a given Merkle commitment.

Game $\mathbf{G}_4$: 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 as follows:

$$(\bar{\mathsf{com}}_{\mathsf{r},1}, \ldots, \bar{\mathsf{com}}_{\mathsf{r},K}) \leftarrow \mathsf{ExtLeafs}_\mathsf{r}(\mathsf{com}_\mathsf{r}, K).$$

Then, the game aborts if there is an instance $i \in [K]$ such that $\bar{\mathsf{com}}_{\mathsf{r},i} = \ \perp$ but later algorithm Check outputs 1. Recall that algorithm Check verifies that

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

Thus, for every fixed interaction, we can bound the probability of such an abort using a straight-forward reduction from the game in Lemma 3. Using a union bound we obtain

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

Game $\mathbf{G}_5$: We introduce another abort in the signer oracle. In this game, after the extraction $(\bar{\mathsf{com}}_{\mathsf{r},1}, \ldots, \bar{\mathsf{com}}_{\mathsf{r},K}) \leftarrow \mathsf{ExtLeafs}_\mathsf{r}(\mathsf{com}_\mathsf{r}, K)$ we introduced before, the game extracts

$$(\bar{\mathsf{r}}_{i,1}, \ldots, \bar{\mathsf{r}}_{i,N}) \leftarrow \mathsf{ExtLeafs}_\mathsf{r}(\mathsf{com}_{\mathsf{r},i}, N)$$

for every $i \in [K]$ for which $\bar{\mathsf{com}}_{\mathsf{r},i} \neq \ \perp$. 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. For each fixed interaction and each instance $i \in [K]$, we can bound the probability of such an abort by a reduction from the game in Lemma 4. By a union bound we get

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

Game $\mathbf{G}_6$: 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

$$(\bar{c}_{1,1}, \ldots, \bar{c}_{K,N}) \leftarrow \mathsf{ExtLeafs}_c(\mathsf{com}_c, KN).$$

If there is an index $(i,j) \in [K] \times [N]$ such that $\bar{c}_{i,j} = \ \perp$ but later algorithm Check outputs 1, the game aborts. Note that algorithm Check internally checks if

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

Thus, for each fixed interaction it is possible to construct a straight-forward reduction from the game in Lemma 3 to bound the probability of such an abort and hence we obtain

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

Game $\mathbf{G}_7$: In $\mathbf{G}_7$, 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 $\mathbf{G}_6$ and $\mathbf{G}_7$ 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

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

Game $\mathbf{G}_8$: In $\mathbf{G}_8$, 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 $\mathbf{G}_7$. Further, $\mathsf{seed}_{\mathbf{J}}$ is sampled uniformly at random. A union bound over all $Q_{\mathsf{H}'}$ queries and q interactions shows that

$$|\Pr[\mathbf{G}_7 \Rightarrow 1] - \Pr[\mathbf{G}_8 \Rightarrow 1]| \leq \frac{qQ_{\mathsf{H}'}}{2^n}.$$

Now, fix an interaction in $\mathbf{G}_8$ 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 6), 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 $\mathbf{G}_9$: Game $\mathbf{G}_9$ is defined as $\mathbf{G}_8$, but additionally aborts if for all $i \in [K]$ we have $\mathsf{cheat}_i^* \geq \frac{1}{\vartheta} + \delta$. In particular, if $\mathbf{G}_9$ 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 $\mathbf{G}_8$ and $\mathbf{G}_9$ 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 $\mathsf{cheat}_i^* \geq \frac{1}{\vartheta} + \delta$. We note that the coins ρ that the experiment uses for the K instances are independent. Thus we have

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

which implies

$$\left|\Pr\left[\mathbf{G}_8 \Rightarrow 1\right] - \Pr\left[\mathbf{G}_9 \Rightarrow 1\right]\right| \leq \Pr_{\rho,\rho_{\mathcal{A}}}\left[\mathsf{bad}\right] \leq e^{-\delta K}.$$

Game $\mathbf{G}_{10}$: In game $\mathbf{G}_{10}$, we sample a random instance $i^* \leftarrow\!\!\$\, [K]$ at the beginning of the game. In the end, the game aborts if $\mathsf{cheat}_{i^*}^* \geq \frac{1}{\vartheta} + \delta$. 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

$$\begin{aligned}
\Pr\left[\mathbf{G}_{10} \Rightarrow 1\right] &= \Pr\left[\mathbf{G}_9 \Rightarrow 1 \wedge \mathsf{cheat}^*_{i^*} < \frac{1}{\vartheta} + \delta\right] \\
&= \Pr\left[\mathbf{G}_9 \Rightarrow 1\right] \cdot \Pr\left[\mathsf{cheat}^*_{i^*} < \frac{1}{\vartheta} + \delta \mid \mathbf{G}_9 \Rightarrow 1\right] \\
&\geq \Pr\left[\mathbf{G}_9 \Rightarrow 1\right] \cdot \Pr\left[\mathsf{cheat}^*_{i^*} < \frac{1}{\vartheta} + \delta \mid \exists i \in [K] : \mathsf{cheat}^*_i < \frac{1}{\vartheta} + \delta\right] \\
&\geq \Pr\left[\mathbf{G}_9 \Rightarrow 1\right] \cdot \frac{1}{K},
\end{aligned}$$

where the first inequality follows from the fact that the event $\mathbf{G}_9 \Rightarrow 1$ implies the event $\exists i \in [K] : \mathsf{cheat}^*_i < \frac{1}{\vartheta} + \delta$.

Game $\mathbf{G}_{11}$: In game $\mathbf{G}_{11}$, 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 $\mathbf{G}_{11}$, 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 $\mathbf{G}_{11}$ 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

$$\left|\Pr\left[\mathbf{G}_{10} \Rightarrow 1\right] - \Pr\left[\mathbf{G}_{11} \Rightarrow 1\right]\right| \leq \frac{1}{p}.$$

Game $\mathbf{G}_{12}$: In game $\mathbf{G}_{12}$, 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 $\mathbf{G}_{11}$, 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

$$\begin{aligned}\Pr\left[\mathbf{G}_{12} \Rightarrow 1\right] &= \Pr\left[\mathbf{G}_{11} \Rightarrow 1\right] \cdot \Pr\left[b[\mu^*] = 1 \wedge \forall \mu \in \mathcal{L} : b[\mu] = 0\right] \\ &= \Pr\left[\mathbf{G}_{11} \Rightarrow 1\right] \cdot \frac{1}{q+1}\left(1 - \frac{1}{q+1}\right)^q \\ &= \Pr\left[\mathbf{G}_{11} \Rightarrow 1\right] \cdot \frac{1}{q}\left(1 - \frac{1}{q+1}\right)^{q+1} \\ &\geq \Pr\left[\mathbf{G}_{11} \Rightarrow 1\right] \cdot \frac{1}{4q},\end{aligned}$$

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

$$\Pr\left[\mathbf{G}_{12} \Rightarrow 1\right] \leq \epsilon_{\mathsf{CDH}}.$$

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 $\mathsf{pk} := (\mathbb{G}, g, p, e, \mathsf{pk}_1, \ldots, \mathsf{pk}_K, \mathsf{ck})$ 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 $\mathbf{G}_{12}$. 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 $\mathbf{G}_{12}$. 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 $\mathbf{G}_{12}$ 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 $\mathbf{G}_{12}$, $\mathcal{B}$ also inserts μ into the set $\mathcal{L}$. It computes $s_{i,\mathbf{J}_i}$ for $i \neq i^*$ as game $\mathbf{G}_{12}$ 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 $\mathbf{G}_{12}$. 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 $\mathbf{G}_{12}$, $\mathcal{B}$ 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 $\mathbf{G}_{12}$ 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 $\mathbf{G}_{12}$ 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 $\mathbf{G}_{12}$ would have aborted.

It remains to show that if $\mathbf{G}_{12}$ 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. □

3.3 Concrete Parameters and Efficiency

Let us now discuss concrete parameters for our scheme $\mathsf{BS}_{\mathsf{CDH}}$ 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 2 to end up with a contradiction. If we use Theorem 2 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 [35].

However, note that we can only apply this approach, if $(1-\delta)\vartheta > 1$, due to Theorem 2. By our choice of δ it is therefore sufficient to guarantee that

$$\left(1 - \frac{\ln(2^{\kappa+1})}{K}\right)\vartheta > 1.$$

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 1 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 entire approach discussed above in simple Python script, see Supplementary Material Section J.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. Also, we 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.

4 Scheme based on OGQ

In this section we construct a blind signature scheme based on the RSA assumption. Our scheme is based on the Okamoto-Guillou-Quisquater (OGQ) [28] 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 [21] 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 [21].

Further, 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).

We show that Invert outputs properly distributed preimages in Supplementary Material Section G.1.

4.1 Construction

Here, we present our construction $\mathsf{BS}_{\mathsf{RSA}} = (\mathsf{Gen}, \mathsf{S}, \mathsf{U}, \mathsf{Ver})$, 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 C). 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 $\mathsf{H}_\mathsf{r}, \mathsf{H}_c : \{0,1\}^* \to \{0,1\}^n, \mathsf{H}_x : \{0,1\}^* \to \mathcal{D} \times \mathbb{Z}_\lambda \times \mathcal{R}_{\mathsf{ck}} \times \{0,1\}^{n_{\mathsf{PRF}}}$.

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 3 and 4.

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** $\mathsf{com}_\mathsf{r} \neq \mathsf{ptree}_I^{\mathsf{H}_\mathsf{r}}(\mathsf{r}_1, \ldots, \mathsf{r}_{J-1}, \eta, \mathsf{r}_{J+1}, \ldots, \mathsf{r}_N)$: **return** 0
8 : **if** $\mathsf{com}_c \neq \mathsf{tree}^{\mathsf{H}_c}(c_1, \ldots, c_N)$: **return** 0
9 : **return** 1

Fig. 3. The algorithm Check used in the issuing protocol of blind signature scheme $\mathsf{BS}_{\mathsf{RSA}}$, 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.

4.2 Security Analysis

Completeness of the scheme is immediate. We show blindness and one-more unforgeability.

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

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

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

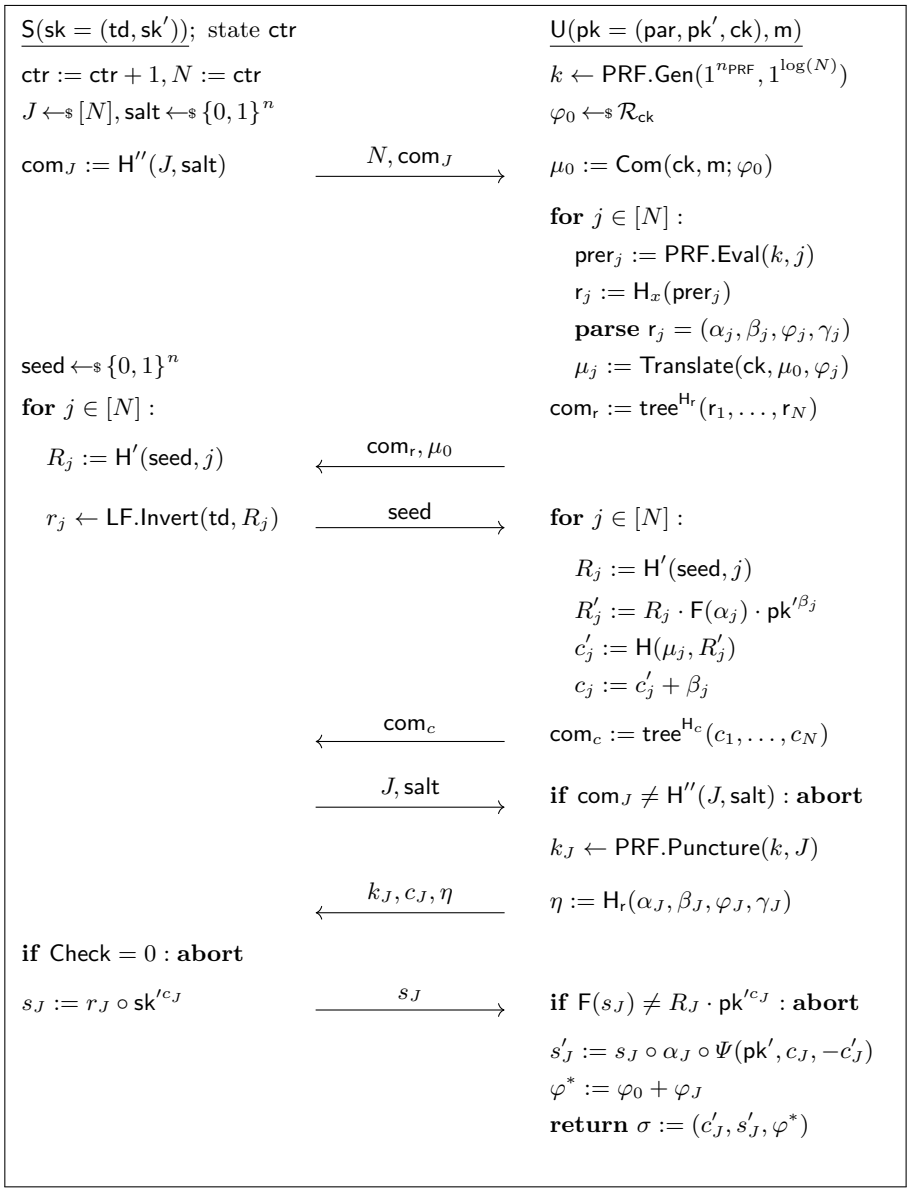

Fig. 4. The signature issuing protocol of the blind signature scheme $\mathsf{BS}_{\mathsf{RSA}}$, where $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_\lambda, \mathsf{H}' : \{0,1\}^* \to \mathbb{Z}_N^*, \mathsf{H}'' : \{0,1\}^* \to \{0,1\}^n$ and $\mathsf{H}_r, \mathsf{H}_c : \{0,1\}^* \to \{0,1\}^n, \mathsf{H}_x : \{0,1\}^* \to \mathcal{D} \times \mathbb{Z}_\lambda \times \mathcal{R}_{\mathsf{ck}} \times \{0,1\}^{n_{\mathsf{PRF}}}$ are random oracles. The algorithm Check is defined in Figure 3. The state ctr of S is atomically incremented at the beginning of every interaction.

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.*

Due to space limitation, we postpone the proof to Supplementary Material Section G.4.

Theorem 4. *Let* PRF *be a puncturable pseudorandom function and* CMT *be a randomness homomorphic commitment scheme. Further, let* $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_\lambda, \mathsf{H}' : \{0,1\}^* \to \mathbb{Z}_N^*, \mathsf{H}'' : \{0,1\}^* \to \{0,1\}^n$ *and* $\mathsf{H}_\mathsf{r}, \mathsf{H}_c : \{0,1\}^* \to \{0,1\}^n$ *be random oracles. Then* $\mathsf{BS}_{\mathsf{RSA}}$ *satisfies one-more unforgeability.*

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* $\mathsf{H}_\mathsf{r}, \mathsf{H}_c, \mathsf{H}', \mathsf{H}'', \mathsf{H}$*, 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* $\mathsf{CCBS}[\mathsf{CMT}]$ *(cf. Supplementary Material Section G.2) 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* $\epsilon_{\mathsf{CCBS}[\mathsf{CMT}]}$ *such that*

$$\epsilon \leq \frac{Q_{\mathsf{H}_\mathsf{r}}^2}{2^{n-1}} + \frac{Q_{\mathsf{H}_c}^2}{2^{n-1}} + \frac{pQ_{\mathsf{H}_\mathsf{r}}}{2^n} + \frac{pQ_{\mathsf{H}_c}}{2^n} + \frac{pQ_{\mathsf{H}'}}{2^n} + \frac{pQ_{\mathsf{H}''}}{2^n} + \epsilon_{\mathsf{CCBS}[\mathsf{CMT}]}.$$

We postpone the formal proof to Supplementary Material Section G.3. On a high level, it mimics the proof of Theorem 2 with $K = 1$ and a reduction from the original boosting transform.

4.3 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 4 and 5. 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 4 and 5 we obtain that

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

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 [10], which is similar to [21]. 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[3] 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 3, the blindness advantage can be upper bounded by

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

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

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

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

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

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. 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
9. 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)
10. Crandall, R., Pomerance, C.B.: Prime numbers: a computational perspective, vol. 182. Springer Science & Business Media (2006)
11. 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)
12. 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)
13. 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)
14. 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)

15. 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)
16. 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)
17. 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)
18. 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)
19. 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)
20. 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)
21. 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)
22. 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)
23. 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)
24. 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)
25. 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)
26. Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) Advances in Cryptology – CRYPTO'89. Lecture Notes in Computer Science, vol. 435, pp. 218–238. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 1990)
27. NIST: Digital signature standard (dss). FIPS PUB 186-4, Federal Information Processing Stnadards Publication (2013), https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

28. 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)
29. 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)
30. 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)
31. 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)
32. 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)
33. Pointcheval, D., Stern, J.: Provably secure blind signature schemes. In: Kim, K., Matsumoto, T. (eds.) Advances in Cryptology – ASIACRYPT'96. Lecture Notes in Computer Science, vol. 1163, pp. 252–265. Springer, Heidelberg, Germany, Kyongju, Korea (Nov 3–7, 1996)
34. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. Journal of Cryptology 13(3), 361–396 (Jun 2000)
35. Pollard, J.M.: Monte carlo methods for index computation mod p. Mathematics of computation 32(143), 918–924 (1978)
36. 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)

Supplementary Material

A Standard Computational Assumptions

Definition 6 (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:*

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

Definition 7 (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:*

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

B Merkle Trees

Our construction makes use of Merkle hash trees [26] instantiated with a random oracle.

Definition 8 (Merkle Tree Evaluation). *For a random oracle* $\mathsf{H} : \{0,1\}^* \to \{0,1\}^n$ *we define Merkle tree evaluation as follows:*

$$\begin{aligned} \mathsf{tree}^{\mathsf{H}}(x_1) &:= \mathsf{H}(x_1), \\ \mathsf{tree}^{\mathsf{H}}(x_1, \ldots, x_\ell) &:= \mathsf{H}(\mathsf{tree}(x_1, \ldots, x_{\lceil \ell/2 \rceil}), \mathsf{tree}(x_{\lceil \ell/2 \rceil + 1}, \ldots, x_\ell)), \end{aligned}$$

where $\ell \in \mathbb{N}$ *and* $x_1, \ldots, x_\ell \in \{0,1\}^*$.

We also define the pruned evaluation of a Merkle hash tree. That is, the evaluation if for one leaf x_I only the hash $\mathsf{H}(x_I)$ is given.

Definition 9 (Pruned Merkle Tree). *Let* $\mathsf{H} : \{0,1\}^* \to \{0,1\}^n$ *be a random oracle. For* $h \in \{0,1\}^n$ *we define* $\mathsf{ptree}_1^{\mathsf{H}}(h) := h$. *Further, let* $\ell \geq 2$ *and* $x_1, \ldots, x_\ell \in \{0,1\}^*$. *For* $I \leq \lceil \ell/2 \rceil$ *we define*

$$\mathsf{ptree}_I^{\mathsf{H}}(x_1, \ldots, x_\ell) := \mathsf{H}(\mathsf{ptree}_I^{\mathsf{H}}(x_1, \ldots, x_{\lceil \ell/2 \rceil}), \mathsf{tree}^{\mathsf{H}}(x_{\lceil \ell/2 \rceil + 1}, \ldots, x_\ell))$$

and for $\lceil \ell/2 \rceil < I \leq \ell$ *we define*

$$\mathsf{ptree}_I^{\mathsf{H}}(x_1, \ldots, x_\ell) := \mathsf{H}(\mathsf{tree}^{\mathsf{H}}(x_1, \ldots, x_{\lceil \ell/2 \rceil}), \mathsf{ptree}_{I - \lceil \ell/2 \rceil}^{\mathsf{H}}(x_{\lceil \ell/2 \rceil + 1}, \ldots, x_\ell)).$$

From these definitions, the following statement follows directly by inspection. Looking ahead, it will imply completeness of our constructions.

Lemma 1. *Let* $\mathsf{H} : \{0,1\}^* \to \{0,1\}^n$ *be a random oracle,* $\ell \in \mathbb{N}$ *and* $x_1, \ldots, x_\ell \in \{0,1\}^*$ *be arbitrary strings. Then for each* $I \in [\ell]$ *the following holds:*

$$\mathsf{tree}^{\mathsf{H}}(x_1, \ldots, x_\ell) = \mathsf{ptree}_I^{\mathsf{H}}(x_1, \ldots, x_{I-1}, \mathsf{H}(x_I), x_{I+1}, \ldots, x_\ell).$$

We show how to extract from Merkle commitments. That is, suppose a reduction can observe the random oracle that is used to commit to a set of values using Merkle trees. Then we show that the reduction can (almost) always extract the values from their commitment. In Figure 5 we define an algorithm ExtLeafs that realizes this extraction. We now present a sequence of lemmas that capture the basic properties of this extraction. The first lemma follows trivially by inspection and the definition of Merkle trees. It says that ExtLeafs is the inverse operation for $\mathsf{tree}^{\mathsf{H}}$, provided that it successfully extracts all leafs.

$\mathsf{ExtLeafs}(\mathsf{com}, \ell)$

1 : **if** $h^{-1}[\mathsf{com}] = \ \perp$:
2 : $\quad(\mathsf{pre}_1, \ldots, \mathsf{pre}_\ell) := (\perp, \ldots, \perp)$
3 : $\quad$**return** $(\mathsf{pre}_1, \ldots, \mathsf{pre}_\ell)$
4 : **if** $\ell = 1$: **return** $h^{-1}[\mathsf{com}]$
5 : **parse** $h^{-1}[\mathsf{com}] = (\mathsf{com}_0, \mathsf{com}_1)$
6 : $\mathsf{pre}_0 := \mathsf{ExtLeafs}(\mathsf{com}_0, \lceil \ell/2 \rceil)$
7 : $\mathsf{pre}_1 := \mathsf{ExtLeafs}(\mathsf{com}_1, \ell - \lceil \ell/2 \rceil)$
8 : **return** $(\mathsf{pre}_0, \mathsf{pre}_1)$

$\mathsf{H}(x)$

1 : **if** $h[x] = \ \perp$:
2 : $\quad y \leftarrow\!\!\$\ \{0,1\}^n$
3 : $\quad$**if** $\exists x' : h[x'] = y$:
4 : $\quad\quad$**return** $\perp$
5 : $\quad$**if** $h[y] \neq \ \perp$:
6 : $\quad\quad$**return** $\perp$
7 : $\quad h[x] := y, h^{-1}[y] := x$
8 : **return** $h[x]$

Fig. 5. Algorithm $\mathsf{ExtLeafs}$ and random oracle $\mathsf{H} : \{0,1\}^* \to \{0,1\}^n$. In the proof of Theorems 2 and 4, the random oracles $\mathsf{H}_\mathsf{r}, \mathsf{H}_c$ are simulates as it is presented here.

Lemma 2. *Let $N = \mathsf{poly}(n)$ be a natural number. Consider an oracle $\mathsf{H} : \{0,1\}^* \to \{0,1\}^n$ and algorithm $\mathsf{ExtLeafs}$ as defined in Figure 5. Let $\mathsf{com} \in \{0,1\}^n$ and assume that the output $(\mathsf{pre}_1, \ldots, \mathsf{pre}_N)$ of $\mathsf{ExtLeafs}(\mathsf{com}, N)$ satisfies that there is no $i \in [N]$ such that $\mathsf{pre}_i = \ \perp$. Then $\mathsf{tree}^\mathsf{H}(\mathsf{pre}_1, \ldots, \mathsf{pre}_N) = \mathsf{com}$.*

Next, we show the following two simple lemmas, which state that if the algorithm $\mathsf{ExtLeafs}$ can not extract all leafs from a given Merkle commitment, then it is statistically hard for an adversary to open it afterwards.

Lemma 3. *Let $N = \mathsf{poly}(n)$ be a natural number. Consider an oracle $\mathsf{H} : \{0,1\}^* \to \{0,1\}^n$ and algorithm $\mathsf{ExtLeafs}$ as defined in Figure 5. Further, for any adversary $\mathcal{A}$ consider the following game:*

1. *Run $(\mathsf{com}, St) \leftarrow \mathcal{A}^\mathsf{H}(1^n)$.*
2. *Run $(\mathsf{pre}_1, \ldots, \mathsf{pre}_N) \leftarrow \mathsf{ExtLeafs}(\mathsf{com}, N)$ and let*

$$L := \{i \in [N] \mid \mathsf{pre}_i = \ \perp\}$$

   be the set of indices of leafs that cannot be extracted from com.
3. *Run $(x_1, \ldots, x_N) \leftarrow \mathcal{A}^\mathsf{H}(St)$.*
4. *Output 1 if and only if $L \neq \emptyset$ and $\mathsf{com} = \mathsf{tree}^\mathsf{H}(x_1, \ldots, x_N)$.*

Denote the number of queries to H made by $\mathcal{A}$ by Q. Then the probability that the game outputs 1 is at most $Q/2^n$.

Proof. We prove the statement via (strong) induction over N. For $N = 1$, the game outputs 1 if and only if $\mathsf{H}(x_1) = \mathsf{com}$, but in step 2 we have $h^{-1}(\mathsf{com}) = \ \perp$. As com is fixed after step 1, for each of the following queries to H the resulting image is equal to com with probability at most 2^{-n}. A union bound leads to the desired bound.

Now, let $N > 1$. The induction hypothesis is that the statement holds for all $N' < N$. We distinguish two cases.

In the first case, we assume that $h^{-1}(\mathsf{com}) = \quad \perp$ in step 2. Here an analogous argument as above is sufficient.

In the second case, we have $h^{-1}(\mathsf{com}) = (\mathsf{com}_0, \mathsf{com}_1)$ and $L \neq \emptyset$ in step 2. Define

$$L_0 := L \cap [\lceil \frac{N}{2} \rceil], \quad L_1 := L \setminus L_0.$$

Clearly, at least one of these two sets is not empty. Without loss of generality, assume that $L_0 \neq \emptyset$. Denote the event that we are in this case and the game outputs 1 by bad. Then we can apply the induction hypothesis to $\lceil N/2 \rceil, \mathsf{com}_0, L_0$ and $x_1, \ldots, x_{\lceil N/2 \rceil}$ to bound the probability of bad. The claim follows.

To be more precise, we can construct a reduction that wins the game for $N' := \lceil N/2 \rceil$ if the game outputs 1 in the case specified above. The reduction is as follows: It gets as input the parameter 1^n and access to an oracle H. It runs $\mathcal{A}$ with access to H and obtains a commitment com. Note that the reduction forwards queries $\mathsf{H}(x)$ from $\mathcal{A}$ to its own game and can keep track of the associative array $h^{-1}[\cdot]$ that its own game holds. Then, $\mathcal{A}$ outputs $x_1, \ldots, x_N$, and the reduction retrieves $h^{-1}(\mathsf{com}) = (\mathsf{com}_0, \mathsf{com}_1)$, which is defined by assumption. It outputs com_0 to its own game. Then, it outputs $x_1, \ldots, x_{\lceil N/2 \rceil}$ to its own game. Clearly, if bad occurs, then the reduction wins its own game. Thus, the induction hypothesis implies that the probability of bad is at most $Q/2^n$. □

Lemma 4. *Let $N = \mathsf{poly}(n)$ be a natural number. Consider an oracle $\mathsf{H} : \{0,1\}^* \to \{0,1\}^n$ and algorithm $\mathsf{ExtLeafs}$ as defined in Figure 5. Further, for any adversary $\mathcal{A}$ consider the following game:*

1. *Run $(\mathsf{com}, St) \leftarrow \mathcal{A}^{\mathsf{H}}(1^n)$.*
2. *Run $(\mathsf{pre}_1, \ldots, \mathsf{pre}_N) \leftarrow \mathsf{ExtLeafs}(\mathsf{com}, N)$ and let*

$$L := \{i \in [N] \mid \mathsf{pre}_i = \quad \perp\}$$

   be the set of indices of leafs that cannot be extracted from com.
3. *Sample $J \leftarrow_s [N]$ and run $(x_1, \ldots, x_{J-1}, \eta, x_{J+1}, x_N) \leftarrow \mathcal{A}^{\mathsf{H}}(St, J)$.*
4. *Output 1 if and only if $L \setminus \{J\} \neq \emptyset$ and $\mathsf{com} = \mathsf{ptree}^{\mathsf{H}}_J(x_1, \ldots, x_{J-1}, \eta, x_{I+1}, \ldots, x_N)$.*

Denote the number of queries to H made by $\mathcal{A}$ by Q. Then the probability that the game outputs 1 is at most $Q/2^n$.

Proof. We prove the statement via (strong) induction over N. For $N = 1$, the statement holds trivially, as $L \setminus \{J\}$ is always the empty set.

For $N > 1$ we distinguish two cases. The induction hypothesis is that the statement holds for all $N' < N$.

First, if $h^{-1}(\mathsf{com}) = \quad \perp$ in step 2, then for each of the the queries $\mathsf{H}(x)$ made by $\mathcal{A}$ after step 1 the probability that the hash value is equal to com is at most 2^{-n}. Hence, a union bound leads to the desired upper bound.

The second case we consider is that $h^{-1}(\mathsf{com}) = (\mathsf{com}_0, \mathsf{com}_1)$. Define

$$L_0 := L \cap [\lceil \frac{N}{2} \rceil], \quad L_1 := L \setminus L_0.$$

Without loss of generality, we assume $J \in [\lceil N/2 \rceil]$. Then, if $L_1 \neq \emptyset$, the claim follows by applying Lemma 3 to $N - \lceil N/2 \rceil, \mathsf{com}_1, L_1$ and $x_{\lceil N/2 \rceil + 1}, \ldots, x_N$. To be more precise, we construct a reduction that wins the game in Lemma 3 for $N' := N - \lceil N/2 \rceil$. The reduction has access to an oracle H and runs $\mathcal{A}$ by forwarding queries from $\mathcal{A}$ to H. Thereby it can also keep track of the associative array $h^{-1}[\cdot]$. When $\mathcal{A}$ outputs com, it retrieves $h^{-1}(\mathsf{com}) = (\mathsf{com}_0, \mathsf{com}_1)$ and outputs com_1 to its own game. Then, when $\mathcal{A}$ outputs $x_1, \ldots, X_N$, it outputs $x_{\lceil N/2 \rceil + 1}, \ldots, x_N$ to its own game.

Otherwise, if $L_1 = \emptyset$, then $L_0 = L$ and $L_0 \setminus \{J\} \neq \emptyset$. Thus, we can apply the induction hypothesis to $\lceil N/2 \rceil, \mathsf{com}_0, L_0$ and $x_1, \ldots, x_{J-1}, \eta, x_{J+1}, \ldots, x_{\lceil N/2 \rceil}$. Again, this can be done more formally by providing a reduction similar to the one discussed above. □

C 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 [20]. 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 Randomness Homomorphic Commitment from DLOG

We also show that the standard Pedersen commitment scheme [31] 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:

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

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 [18]. 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 5. *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 $x = x_1 \parallel \bar{x}$ 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 6. *For a sum X of independent $\{0,1\}$-random variables and any $s > \mu := \mathbb{E}[X]$ it holds that*

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

Proof. The proof is similar to [25]. 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 Ommitted Analysis of Our Scheme from RSA

In this section we formaly analyze our RSA-based Scheme, which we ommitted in the main body.

G.1 Invertibility of the OGQ Linear Function

Recall the definition of algorithm $\mathsf{Invert}(\mathsf{td}, z)$ for $z \in \mathbb{Z}_N^*$:

- 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).

Here, we show 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

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

and

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

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.

G.2 The Boosting Transform

We revisit the boosting transform introduced in [25] 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 [25] 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 [21,25].

Theorem 5. *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* $\ell = 3\ln(p+1) + \ln(2/\epsilon)$.

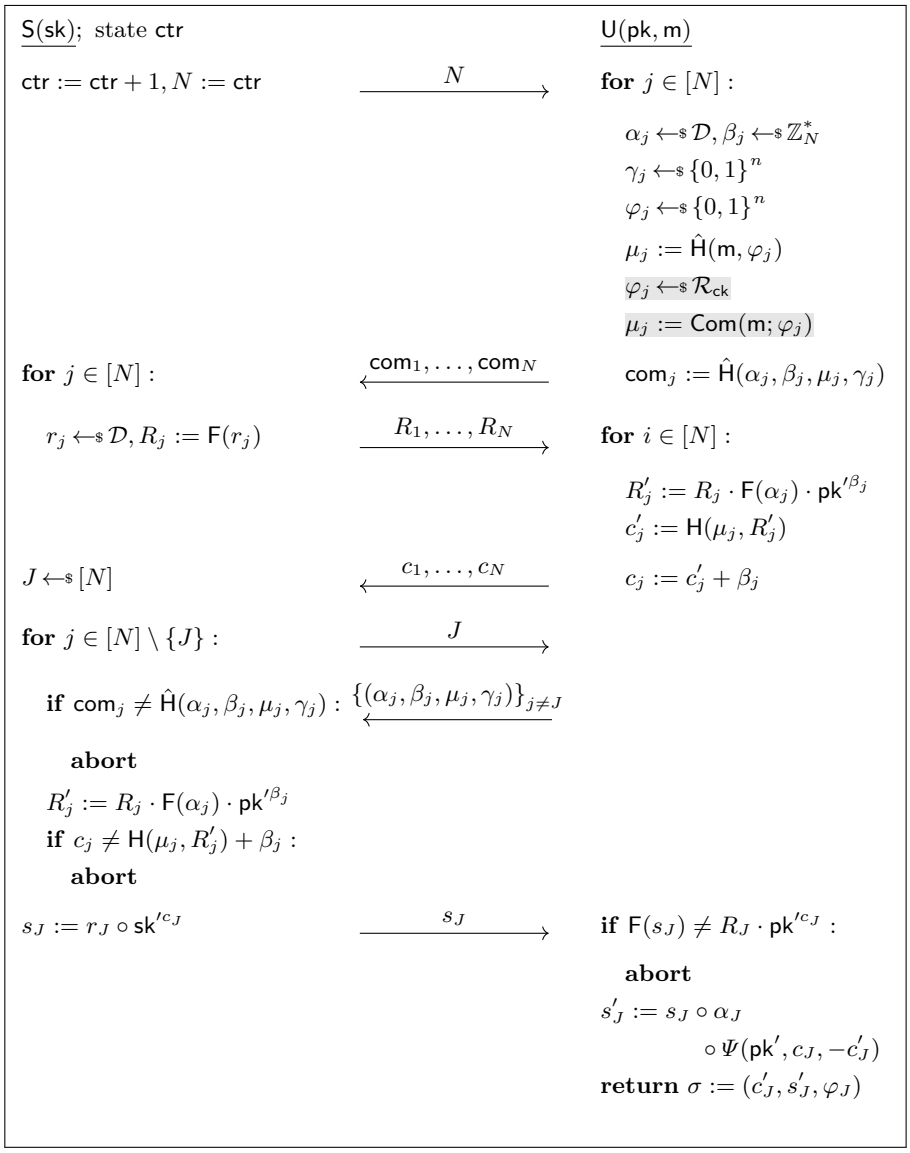

Fig. 6. The signature issuing protocol of the blind signature scheme CCBS obtained via the boosting construction, where $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_\lambda, \hat{\mathsf{H}} : \{0,1\}^* \to \{0,1\}^n$ are random oracles. The state ctr of S is atomically incremented at the beginning of every 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.

G.3 One-More Unforgeability

Proof (of Theorem 4). 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 2.

Game $\mathbf{G}_0$: We start with game $\mathbf{G}_0 := \mathbf{OMUF}^{\mathcal{A}}_{\mathsf{BS}}$, 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 $\mathsf{H}, \mathsf{H}', \mathsf{H}_{\mathsf{r}}$ 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 $\mathbf{G}_1$: 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 $Q^2_{\mathsf{H}_t}$ pairs of random oracle queries and obtain

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

Game $\mathbf{G}_2$: In game $\mathbf{G}_2$ we add another change to the random oracles $\mathsf{H}_t, t \in \{\mathsf{r}, c\}$. We again sample $y \leftarrow\!\!\$ \{0,1\}^n$ if $\mathsf{H}_t(x)$ is not yet defined. This time, we also check if $\mathsf{H}_t(y)$ is already defined and return $\perp$ if this is the case. The detailed behavior of oracle H_t can be found in Figure 5. Note that the adversary $\mathcal{A}$ can only distinguish between $\mathbf{G}_1$ and $\mathbf{G}_2$ if such a chain happens, i.e. H_t returns $\perp$ because $\mathsf{H}_t(y)$ was already defined. Again, we can apply a union bound over all $Q^2_{\mathsf{H}_t}$ pairs of random oracle queries and obtain

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

To summarize the changes we did so far, we ruled out two patterns for random oracle queries:

1. Collisions: $\exists x \neq x' : \mathsf{H}_t(x) = \mathsf{H}_t(x')$.
2. Chains: $\exists x$: query $\mathsf{H}_t(\mathsf{H}_t(x))$ was made before query $\mathsf{H}_t(x)$.

In particular, this 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. Further,

note that for any $x_0 \in \{0,1\}^n$ the sequence $(x_i)_i$ with $x_i := \mathsf{H}_t^{-1}(x_{i-1})$ for $i \in \mathbb{N}$ does contain any repeating value. Indeed, such a cycle could only be produced if the adversary managed to form a chain, which was ruled out. Note that this implies that algorithm $\mathsf{ExtLeafs}_t$ given in Figure 5 always terminates.

Game $\mathbf{G}_3$: In game $\mathbf{G}_3$, we change the way the signing oracle is executed. Whenever $\mathcal{A}$ sends $\mathsf{com}_\mathsf{r}, \mu_0$ as its first message, the game tries to extract the messages from this commitment using algorithm $\mathsf{ExtLeafs}_\mathsf{r}$, i.e. it runs

$$(\bar{\mathsf{r}}_1, \ldots, \bar{\mathsf{r}}_N) \leftarrow \mathsf{ExtLeafs}_\mathsf{r}(\mathsf{com}_\mathsf{r}, N).$$

If there is a session $j \in [N]$ such that $\bar{\mathsf{r}}_j = \ \perp$, i.e. the algorithm $\mathsf{ExtLeafs}_\mathsf{r}$ 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 $\mathbf{G}_2$ and $\mathbf{G}_3$. For each fixed interaction, we can bound the probability of this event (with respect to that interaction) by a straight forward reduction from the game in Lemma 4. By a union bound we obtain

$$|\Pr[\mathbf{G}_2 \Rightarrow 1] - \Pr[\mathbf{G}_3 \Rightarrow 1]| \leq \Pr[\mathsf{bad}_1] \leq \frac{pQ_{\mathsf{H}_\mathsf{r}}}{2^n}.$$

Game $\mathbf{G}_4$: 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 runs

$$(\bar{c}_1, \ldots, \bar{c}_N) \leftarrow \mathsf{ExtLeafs}_c(\mathsf{com}_c, N).$$

That is, it tries to extract the leafs from the commitment com_c using the random oracle H_c via algorithm $\mathsf{ExtLeafs}_c$. Then, if there is an index $j \in [N]$ such that $\bar{c}_j = \ \perp$, i.e. the game was 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

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

Thus, it is possible to construct a straight forward reduction from the game in Lemma 3 to bound the probability of bad_2 in a fix interaction and hence the distinguishing advantage of $\mathcal{A}$ between $\mathbf{G}_3$ and $\mathbf{G}_4$. We obtain

$$|\Pr[\mathbf{G}_3 \Rightarrow 1] - \Pr[\mathbf{G}_4 \Rightarrow 1]| \leq \Pr[\mathsf{bad}_2] \leq \frac{pQ_{\mathsf{H}_c}}{2^n}.$$

Game $\mathbf{G}_5$: 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

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

Game $\mathbf{G}_6$: In $\mathbf{G}_6$, 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 $\mathbf{G}_5$ and $\mathbf{G}_6$ 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

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

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 $\mathbf{G}_6$ 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 $\mathbf{G}_6$ 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 $\mathbf{G}_6$.
- 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 via $(\bar{\mathsf{r}}_1, \ldots, \bar{\mathsf{r}}_N) \leftarrow \mathsf{ExtLeafs}_\mathsf{r}(\bar{\mathsf{com}}_{\mathsf{r},i}, N)$. 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 $\bar{c}_i$ via $(\bar{c}_1, \ldots, \bar{c}_N) \leftarrow \mathsf{ExtLeafs}_c(\mathsf{com}_c, N)$. For each $j \in [N]$ for which $\bar{c}_j$ is defined, it sets $c'_j := \bar{c}_j$. For the remaining j, 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 $\mathbb{Z}_N^*$ [4] 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. □

G.4 Blindness

Lemma 7. *For any algorithm $\mathcal{A}$ and bit $b \in \{0,1\}$, we consider the following game $\boldsymbol{G}_b$:*

[4] This property is called smoothness in [21].

1. *Sample parameters* $\mathsf{par} = (N, a, \lambda)$ *and a trapdoor* $\mathsf{td} = (p, q)$ *as above. Sample* $\mathsf{sk}' \leftarrow_s \mathcal{D}$*, set* $\mathsf{pk}' := \mathsf{F}(\mathsf{sk}')$*. Let* $\mathsf{H} : \{0,1\}^* \to \mathbb{Z}_\lambda$ *be a random oracle. Run* $(\mathsf{m}_0, \mathsf{m}_1, St) \leftarrow \mathcal{A}^{\mathsf{H}}(\mathsf{par}, \mathsf{pk}', \mathsf{td}, \mathsf{sk}')$*.*
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* $\sigma_1 = \ \perp$*, then 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 $\mathcal{A}$ *that makes at most* Q_{H} *many queries to* H *we have*

$$|\Pr[\boldsymbol{G}_0 \Rightarrow 1] - \Pr[\boldsymbol{G}_1 \Rightarrow 1]| \leq \frac{2Q_{\mathsf{H}}}{\mathbb{Z}_N^*}.$$

Proof. This is a direct application of Theorem 5.8 in the full version of [21]. To be more precise, [21] show that the distinguishing advantage of any adversary is zero as long as it never queries $\mathsf{H}(\cdot, R')$, where R' is the value that the oracle $\mathrm{O}_{b'}$ inputs into H in step 2a. Note that R' is uniform over $\mathbb{Z}_N^*$ as $\mathsf{F}(\alpha)$ is. Therefore, a union bound implies that the probability that the adversary queries $\mathsf{H}(\cdot, R')$ is at most $Q_{\mathsf{H}}/\mathbb{Z}_N^*$ per oracle. □

Proof (of Theorem 3). 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

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

We will show the statement via a sequence of games. Unless otherwise stated, random oracles are simulated honestly via lazy sampling.

Game $\mathbf{G}_{0,b}$: Game $\mathbf{G}_{0,b}$ is defined as the real blindness game $\mathbf{BLIND}^{\mathcal{A}}_{b,\mathsf{BS}}$. Recall that the game first samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^n)$ and obtains two messages $\mathsf{m}_0, \mathsf{m}_1$ from the adversary $\mathcal{A}$. 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 execution 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

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

Game $\mathbf{G}_{1,b}$: Game $\mathbf{G}_{1,b}$ is exactly as game $\mathbf{G}_{0,b}$, 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

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

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

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

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 $\mathsf{prer}^L_{\hat{J}^L}$. 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:

- Sample $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^n)$ and run $(\mathsf{m}_0, \mathsf{m}_1, St) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{sk})$.
- 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 $\mathsf{H}_x(\mathsf{prer}_{\hat{\jmath}})$. However, $\mathcal{A}$ obtains no information about $\mathsf{prer}_{\hat{\jmath}}$, 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 different way. In detail, it samples $\eta \leftarrow\!\!\$\, \{0,1\}^n$ and computes the $\mathsf{com}_{\mathsf{r}}$ in a pruned way, i.e.

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

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 $\mathsf{H}_{\mathsf{r}}(\mathsf{r}^X_{\hat{\jmath}X})$ for an $X \in \{L, R\}$. Note that $\mathcal{A}$ obtains no information about $\gamma_{\hat{\jmath}}$ and $\gamma_{\hat{\jmath}}$ 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 $\mu_{\hat{\jmath}}$ is generated. Recall that in $\mathbf{G}_{7,b}$, this is generated as

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

Note that if the game does not stop, then especially $\hat{J} = J$ and $\varphi^* = \varphi_0 + \varphi_{\hat{\jmath}}$. In game $\mathbf{G}_{8,b}$, we sample $\varphi^* \leftarrow\!\!\$\, \mathcal{R}_{\mathsf{ck}}$ and set $\mu_{\hat{\jmath}} := \mathsf{Com}(\mathsf{ck}, \mathsf{m}; \varphi^*)$. 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 $\varphi_{\hat{\jmath}}$. Thus, we have to consider the distribution of the value $\varphi^* = \varphi_0 + \varphi_{\hat{\jmath}}$ conditioned on $k_{\hat{\jmath}}, (\varphi_0 + \varphi_j)_{j \neq \hat{\jmath}}$ and φ_0. This distribution is uniformly random as $\varphi_{\hat{\jmath}}$ 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 [21], which is statistically blind. We showed this in Lemma 7. Thus, we can reduce from the games in Lemma 7 to bound $\mathcal{A}$'s advantage in distinguishing between $\mathbf{G}_{9,0}$ and $\mathbf{G}_{9,1}$. To do so, we construct an unbounded reduction $\mathcal{B}'$ as follows:

- $\mathcal{B}'$ obtains $(\mathsf{par}, \mathsf{pk}', \mathsf{td}, \mathsf{sk}')$ from its own challenger and samples a commitment key $\mathsf{ck} \leftarrow \mathsf{CMT.Gen}(1^n)$. It invokes $(\mathsf{m}_0, \mathsf{m}_1, St) \leftarrow \mathcal{A}((\mathsf{par}, \mathsf{pk}', \mathsf{ck}), (\mathsf{td}, \mathsf{sk}'))$. 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 μ_0^*, μ_1^* and its state to its challenger.
- $\mathcal{B}'$ is executed with access to oracles O_0' and O_1', which simulate the user of the protocol $\mathsf{BS}_{\mathsf{lin}}$. 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 $k \leftarrow \mathsf{PRF.Gen}(1^{n_{\mathsf{PRF}}}, 1^{\log(N)}), k_{\hat{J}} \leftarrow \mathsf{PRF.Puncture}(k, \hat{J})$. Sample randomness $\mathsf{r}_j := \mathsf{PRF.Eval}(k_{\hat{J}}, j)$ 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 $\mathsf{com}_{\mathsf{r}} := \mathsf{ptree}_{\hat{J}}^{\mathsf{H}_{\mathsf{r}}}(\mathsf{r}_1, \ldots, \mathsf{r}_{\hat{J}-1}, \eta, \mathsf{r}_{\hat{J}+1}, \ldots, \mathsf{r}_N)$. Send μ_0 and $\mathsf{com}_{\mathsf{r}}$ 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 $\mathsf{com}_c := \mathsf{tree}^{\mathsf{H}_c}(c_1, \ldots, c_N)$. 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'.
- Obtain signatures $\sigma_0' = (c_0', s_0')$ and $\sigma_1' = (c_1', s_1')$ from the challenger and set

$$\sigma_0 := (c_0', s_0', \varphi_0^*), \quad \sigma_1 := (c_1', s_1', \varphi_1^*).$$

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

It is easy to see that if $\mathcal{B}'$ runs in game $\mathbf{G}_0$ from Lemma 7, then it perfectly simulates game $\mathbf{G}_{9,0}$ for $\mathcal{A}$, and if it runs in game $\mathbf{G}_1$ from Lemma 7 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 7 that $\mathcal{B}'$ has advantage in distinguishing the games $\mathbf{G}_0$ and $\mathbf{G}_1$ at most $2Q_{\mathsf{H}}/|\mathbb{Z}_N^*|$. Hence we have

$$|\Pr[\mathbf{G}_{9,0} \Rightarrow 1] - \Pr[\mathbf{G}_{9,1} \Rightarrow 1]| \leq \frac{2Q_{\mathsf{H}}}{|\mathbb{Z}_N^*|}.$$

The statement follows from an easy calculation. □

H Ommitted Analysis of Our Scheme from CDH

Here, we give the remaining parts formal analysis of our scheme from CDH.

Proof (of Theorem 1). This can be proven in an analogous way to Theorem 3. 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. □

I Concrete Parameters of the Original Boosting Transform

Here we explain how we estimated the concrete efficiency for the boosting transform [25]. Concretely, we consider the Okamoto-Schnorr instantiation [28] 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 [25].

With this in mind, we combine the concrete bounds given in [25] 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 J.3.

J 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 Sections 3.3 and 4.3 and Supplementary Material Section I.

J.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 Section 4.3.

```
#!/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+1) + 2**(2*log_q_hash_c-secpar+1) + 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+1)
                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 = [32]
plambda_lengths_class_c = [12000]



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'))
```

J.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 3.3.

```
#!/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+1)
```

```
        stat_term_b = 2**(2*log_q_hash_c-secpar+1)
        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 = [20,40]

#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'))
```

J.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 I.

```
#!/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 = 40

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("==> 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))
```