On the Impossibility of Round-Optimal Pairing-Free Blind Signatures in the ROM

Marian Dietz[1], Julia Kastner[2], and Stefano Tessaro[3]

[1] Department of Computer Science
ETH Zurich
Zurich, Switzerland
marian.dietz@inf.ethz.ch
[2] Cryptology Group
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
julia.kastner@cwi.nl
[3] Paul G. Allen School of Computer Science & Engineering
University of Washington
Seattle, USA
tessaro@cs.washington.edu

Abstract. Blind signatures play a central role in cryptographic protocols for privacy-preserving authentication and have attracted substantial attention in both theory and practice. A major line of research, dating back to the 1990s, has focused on constructing blind signatures from pairing-free groups. However, all known constructions in this setting require at least *three* moves of interaction between the signer and the user. These schemes treat the underlying group as a black box and rely on the random oracle in their security proofs. While computationally efficient, they suffer from the drawback that the signer must maintain state during a signing session. In contrast, round-optimal solutions are known under other assumptions and structures (e.g., RSA, lattices, and pairings), or via generic transformations such as Fischlin's method (CRYPTO '06), which employ non-black-box techniques.
This paper investigates whether the three-round barrier for pairing-free groups is inherent. We provide the first negative evidence by proving that, in a model combining the Random Oracle Model (ROM) with Maurer's Generic Group Model, no blind signature scheme can be secure if it signs sufficiently long messages while making at most a logarithmic number of random oracle queries. Our lower-bound techniques are novel in that they address the interaction of both models (generic groups and random oracles) simultaneously.

1 Introduction

Blind signatures were initially introduced by Chaum [21] in the context of e-cash [21,23,48] and then used in e-voting [33]. They enable a user to obtain a signature on a message by a signer, without the signer learning what message is being signed, nor which signature has been generated. Blind signatures have recently attracted industry interest as a stepping stone towards implementations of *anonymous tokens* [35], which are used by Cloudflare's PrivacyPass [27,19], proposals for privacy-preserving ad-click measurement by Apple [3] and Meta [1], along with Google Trust Tokens [4]. Blind Signatures are also used in Apple's iCloud Private Relay [2] and Google One VPN Service [5].

The wish for round-optimality. A multitude of constructions of blind signatures from a variety of assumptions have been proposed over the last four decades. Interestingly, however, most practical applications still rely on RSA-based blind signatures [10,11,28,44], despite RSA being largely deprecated in many other contexts—the idea of blinding RSA was in fact already proposed in Chaum's seminal work [21,22]. There are two reasons for this somewhat anachronistic preference. The first is that RSA blind signatures are easily implementable from cryptographic libraries (such as BoringSSL and NSS) available in browsers. The second is that they are *round optimal*, i.e., they require a single

round trip between a user and the issuer. This is beneficial, as the issuer does not need to maintain any state associated with a particular signing session.

Round optimal blind signatures can also be obtained from pairing-friendly groups (e.g., [16,34,40]) and from lattices (e.g., [7,49,15]). The former can be very efficient, but require pairing support and are often avoided in practice, as pairings are not as widely supported by libraries and standards. The latter, while generally post-quantum secure, are still significantly less efficient than pairing- and RSA-based counterparts. Another option is a generic construction by Fischlin [30], based on a commitment scheme, a (standard) signature scheme, and NIZKs, where the signer (non-blindly) signs a commitment to a message to be signed (rather than the message) itself, and the user then produces a proof of knowledge of a signature on a commitment to the message as the actual blind signature. Instantiations are however expensive.

Absent from our discussion above are constructions from *pairing-free* groups, typically obtained from standard elliptic curves. In some sense, these are most attractive, as very efficient signatures (such as Schnorr signatures [51], EdDSA [14], and ECDSA [8]) solely rely on these curves and it is advantageous to come with blind signing protocols for these (or similar) signatures. And furthermore, pairing-free constructions are equally likely to be implementable from existing libraries. Such constructions have been extensively studied (e.g., [24,52,47,50,6,32,37,54,26,20,43,42,17]), yet all of them to date require at least three moves. This not only forces the issuer to maintain state (hence making them less appealing for practice), but it also leads to challenges in proving their concurrent security, such as those surfaced by ROS attacks [52,55,13].

This paper: Round-optimal pairing-free blind signatures. We stress that in principle one *could* obtain a pairing-free blind signature by instantiating Fischlin's construction, but the use of non-black box techniques would increase the computational and communication costs substantially. All aforementioned blind signatures with at least three rounds, in contrast, make black box use of the underlying group and of a hash function, and this makes them very lightweight. We will capture the class of such schemes within a combination of Maurer's generic group model (GGM) [45] and the random oracle model (ROM) [12], which we formalize, and refer to generically as GGM+ROM. The class of blind signatures in the GGM+ROM prevents for example the use of generic zero-knowledge proofs (as required by Fischlin's construction) that represent a signature scheme as a circuit. It does not exclude Σ-protocol type proofs that are fully algebraic. We stress here that we do not rely on Shoup's GGM [53], as it allows for non-algebraic constructions which in particular make use of the generic-group encodings (such as what has been done in [25] to prove security of Schnorr signatures without relying on the random oracle model).

At the core of our paper is the following question:

> *Can we design round-optimal blind signatures from pairing-free groups in the GGM+ROM?*

An informal folklore intuition says that this should not be possible. Namely, the standard known way towards a round optimal construction is to efficiently instantiate Fischlin's construction using an algebraic non-generic NIZK that leverages the algebraic structure of a signature scheme. However, such fully algebraic signature schemes are known not to exist in pairing-free groups, something which is formally verified by a result of Döttling et al. [29]. However, can we verify this intuition formally? This question has remained open so far.

This paper provides a first formal **negative** answer by ruling out any round-optimal pairing-free blind signature in the GGM+ROM under two constraints. The first is that the user and verification make $O(\log \lambda)$ queries to the random oracle, where λ is the security parameter (this restriction does not apply to the signer). The second is that the message space is sufficiently large (i.e., $\omega(\log \lambda)$ bits). Both restrictions apply to all existing constructions of blind signatures making black-box use of the group.

For simplicity, we will present our result with a third restriction, namely we initially do not cover constructions where the random oracle explicitly outputs a group elements, as opposed to a string.

Still, adapting our result to cover these constructions is not very hard, and we briefly highlight the modifications needed to cover this case, too.

Formally, as in prior work [29], our result takes the form of a concrete attack which performs a polynomial number of group operations and random oracle queries, but can otherwise run in exponential time. Our attack breaks the one-more unforgeability of a blind signature. We note that group and random oracle queries (as opposed to actual running time) are the relevant complexity metric in the GGM+ROM, in that all schemes we consider are secure against such attacks (polynomial queries, with unbounded running time). However, a potential avenue to bypass our impossibility result is to introduce additional computational assumptions *unrelated* to the group.

Concurrent work. Concurrent work by Kastner, Tessaro, and Zaverucha [39] considers the complementary question of what is possible when adopting non-black-box techniques, and how efficiently Fischlin's construction can be instantiated from pairing-free curves. To this end, they present a specific instantiation based on (variants of) the Nyberg-Rueppel signature scheme [46]. Their result uses NIZKs about the correct evaluation of a conversion function from group elements to scalars - this is a non-black-box technique that falls outside of our model. This result is complementary to ours, as we show that such (heavier) non-black box techniques are *inherent* for round optimality. In contrast, they show an efficient instantiation of such non-black-box techniques.

Other important related work. Kastner, Nguyen, and Reichle [38] developed a round-optimal pairing-free scheme that combines both an RSA assumption *and* DDH (in a pairing-free group), and hence does not fit within our model. The core structure behind their scheme is in fact RSA based. Katz, Schröder, and Yerukhimovich [41] proved that blind signatures do not exist in the random oracle model, which in turn implies impossibility of black box constructions from one-way functions and permutations. Our techniques differ from theirs—they rely on the intersection-query sampling technique [36,9] used to prove impossibility of key-agreement in the random oracle model. While we share some similarities in some aspects of our proof, it is not clear whether the intersection-query technique as is can help in our setting. (We also note that key-agreement does exist in the GGM+ROM.)

1.1 Technical Overview

Throughout this paper, we use fraktur font (e.g. $\mathfrak{g}$) to emphasize that the object is a group element of a group, whose order is a publicly known prime p. The group's neutral element and a canonical generator are written as $\mathfrak{o}$ and $\mathfrak{1}$, respectively.

The attack by Döttling et al. We first recap the impossibility result from [29] for standard (non-blind) signature schemes in the GGM, which will serve as a starting point for our attack.

Concretely, [29] considers signature schemes in the GGM, where a signature σ must have, w.l.o.g., the form $\sigma = (t, \mathfrak{g}_\sigma)$, where t is a string, and $\mathfrak{g}_\sigma$ is a vector of group elements. Similarly, let us assume the public key has format $\mathsf{vk} = (s, \mathfrak{g}_{\mathsf{vk}})$, where s is a string and $\mathfrak{g}_{\mathsf{vk}}$ is a vector of group elements that must contain the generator $\mathfrak{1}$. Then, [29] makes the simplifying assumption that verification only checks whether the following system of equations is satisfied (a scheme with this assumption is called "purely algebraic"):[4]

$$A \cdot \mathfrak{g}_\sigma = B \cdot \mathfrak{g}_{\mathsf{vk}} \,, \tag{1}$$

where $A = A(s,t,m)$ and $B = B(s,t,m)$ are matrices that depend on s, t, and the signed message m. It is important to appreciate two things: (1) such a system is not necessarily solvable, i.e., given $A(s,t,m)$, $B(s,t,m)$, and $\mathfrak{g}_{\mathsf{vk}}$, it may be the system has no solution, (2) if there is a solution for some s, m, t, then one can compute it efficiently with a small number of group operations as $\mathfrak{g}_\sigma = A^+ \cdot B \cdot \mathfrak{g}_{\mathsf{vk}}$, where A^+ is a weak left-inverse[5] of $A(s,t,m)$. Then, $\sigma = (t, \mathfrak{g}_\sigma)$ is a valid signature for m. A naïve attack could pick a particular message m, iterate over all t's, compute $\mathfrak{g}_\sigma = A^+ \cdot B \cdot \mathfrak{g}_{\mathsf{vk}}$, and

[4] This is *not* without loss of generality, so we will revisit this later on.
[5] i.e., a matrix such that $AA^+A = A$, which can be found efficiently for any A.

check whether (1) is satisfied. As there are potentially exponentially many t's, this would require an exponential number of group operations.

The key technical observation in [29] is a strategy to efficiently find an m and a t for which (1) is solvable. Crucially, efficiency here is measured in terms of group operations, while the actual running time can be exponential. To this end, they observe that each s, m, t defines a vector space of constraints $L(s,t,m) \subseteq \mathbb{Z}_p^{|\mathfrak{g}_{\mathsf{vk}}|}$ such that there exists $\mathfrak{g}_\sigma$ satisfying (1) if and only if $\mathbf{v}^\mathsf{T} \cdot \mathfrak{g}_{\mathsf{vk}} = \mathfrak{o}$ for each $\mathbf{v} \in L(s,t,m)$. (Intuitively, $L(s,t,m)$ denotes a set of constraints that necessarily need to hold if we want to find a valid signature for m with string portion t.) Importantly, $L(s,t,m)$ can be computed efficiently from $A(s,t,m)$ and $B(s,t,m)$ without group operations as $L(s,t,m) = \mathrm{LKer}(A) \cdot B$. Then, we can find a forgery as follows, for a sequence of random messages $m_1, m_2, \ldots$. We start with $L_0 = \{\mathbf{0}\}$, i.e., L_0 is the trivial subspace. Then, at each step i:

1. If there exists t^* such that $L(s,t^*,m_i) \subseteq L_{i-1}$, then we find $\mathfrak{g}_\sigma^*$ satisfying (1) as above. We obtain a forgery $\sigma^* = (t^*, \mathfrak{g}_\sigma^*)$.
2. Otherwise, we ask for a signature on m_i, learning $\sigma_i = (t_i, \mathfrak{g}_{\sigma,i})$. We then set $L_i = L_{i-1} + L(s,t_i,m_i)$, where "+" denotes the vector space sum.

If the attack returns σ^*, this must be a valid signature due to the invariant that each L_i forms a vector space of true constraints that are known to hold for $\mathfrak{g}_{\mathsf{vk}}$.

We note that step 2 can occur at most a finite number of times, corresponding to the dimension of $\mathfrak{g}_{\mathsf{vk}}$. Further, step 1 is generally inefficient (we need to iterate over all t^*), but the check $L(s,t^*,m_i) \subseteq L_{i-1}$ does not need any GGM operations. Therefore, the overall attack requires a polynomial number of group operations.

Extension to general verification. Looking ahead, we will eventually need to handle arbitrary verification algorithms for blind signature schemes in the combined generic group and random oracle model.

As a first step towards this, we have to extend the described framework of [29] beyond the class of purely algebraic signatures. In particular, we allow arbitrary (but deterministic) verification procedures in the GGM. This means that we now allow for group-equality checks that may depend on the outcomes of previous such checks. We can no longer write verification as an explicit relation as in (1) that, once fulfilled, immediately implies a valid signature. A similar generalization was already done in [18], but we will view the generalization from a different angle.

Recall that, in the GGM, for every group-equality $\mathfrak{h} \stackrel{?}{=} 0$ that is checked within $\mathsf{Verify}((s, \mathfrak{g}_{\mathsf{vk}}), m, (t, \mathfrak{g}_\sigma))$, the verification algorithm *knows a representation* $\mathbf{v}$ of the group elements $\mathfrak{h}$ in terms of group elements $\mathfrak{g}_{\mathsf{vk}}$ and $\mathfrak{g}_\sigma$,[6] i.e., it can rewrite "$\mathfrak{h} \stackrel{?}{=} 0$" as shown on the left-hand side in (2).

$$\begin{array}{ccc} \underline{\mathsf{Verify}((s, \mathfrak{g}_{\mathsf{vk}}), m, (t, \mathfrak{g}_\sigma))} & & \underline{\mathsf{Sim}((s, \mathfrak{g}_{\mathsf{vk}}), m, (t, L_{\mathsf{sim}}))} \\ \texttt{//some code here} & & \texttt{//some code here} \\ \mathbf{v}^\mathsf{T} \cdot \begin{pmatrix} \mathfrak{g}_{\mathsf{vk}} \\ \mathfrak{g}_\sigma \end{pmatrix} \stackrel{?}{=} \mathfrak{o} & \rightsquigarrow & \mathbf{v} \stackrel{?}{\in} L_{\mathsf{sim}} \\ \texttt{//more code here} & & \texttt{//more code here} \end{array} \tag{2}$$

The core idea towards proving impossibility in this more general setting is what we refer to as a simulation Sim of the verification procedure Verify, illustrated on the right-hand side of the prior equation. It is meant to be functionally equivalent to the original verification procedure, except that Sim replaces all queries that Verify would make to the GGM for checking group-equalities (which would count towards the attack's costs), by a simulation that is based on some suitably defined vector space $L_{\mathsf{sim}} \subseteq \mathbb{Z}_p^{|\mathfrak{g}_{\mathsf{vk}}|+|\mathfrak{g}_\sigma|}$. This set L_{sim} is meant to cover all linear constraints imposed on group elements $\mathfrak{g}_{\mathsf{vk}}$ and $\mathfrak{g}_\sigma$ that are potentially checked at any time during verification, and is hence a

[6] Readers familiar with the algebraic group model [31] may think of $\mathbf{v}^\mathsf{T}$ as an "algebraic representation" in terms of the group elements $\mathfrak{g}_{\mathsf{vk}}$.

generalization of the two matrices A and B from before. Note that "$\mathbf{v} \stackrel{?}{\in} L_{\mathsf{sim}}$" can be tested without issuing any GGM queries, and thus it is "for free" in our setting.

Adapting the strategy we introduced earlier for the purely algebraic case, we can construct an adversary as follows: We start with $L_0 = \{\mathbf{0}\} \subseteq \mathbb{Z}_p^{|\mathfrak{g}_{\mathsf{vk}}|+|\mathfrak{g}_\sigma|}$. As before, L_i should be thought of as a vector space of linear constraints we have learned about $\mathfrak{g}_{\mathsf{vk}}$ so far. (In fact, we will always have $L_i \subseteq \langle \mathbf{e}_1, \ldots, \mathbf{e}_{|\mathfrak{g}_{\mathsf{vk}}|} \rangle$, despite the vectors themselves being of dimension $|\mathfrak{g}_{\mathsf{vk}}| + |\mathfrak{g}_\sigma|$. This oddity will simplify the adversary description.) Then, at each step i:

1. If there exists (t^*, L_{sim}) with $L_{i-1} = L_{\mathsf{sim}} \cap \langle \mathbf{e}_1, \ldots, \mathbf{e}_{|\mathfrak{g}_{\mathsf{vk}}|} \rangle$, such that the simulation $\mathsf{Sim}((s, \mathfrak{g}_{\mathsf{vk}}), m_i, (t^*, L_{\mathsf{sim}}))$ outputs 1, then we compute matrices A and B with $\mathrm{rowsp}(B \mid -A) = L_{\mathsf{sim}}$. Finally, just like before, we compute $\mathfrak{g}_\sigma^* = A^+ \cdot B \cdot \mathfrak{g}_{\mathsf{vk}}$, and obtain a forgery $\sigma^* = (t^*, \mathfrak{g}_\sigma^*)$.[7]
2. Otherwise, we ask for a signature on m_i, learning $\sigma_i = (t_i, \mathfrak{g}_{\sigma,i})$. We run $\mathsf{Verify}((s, \mathfrak{g}_{\mathsf{vk}}), m, (t, \mathfrak{g}_\sigma))$, while keeping track of a vector space L that contains all vectors $\mathbf{v}$ for which an equality-check (left-hand side in (2)) returns 1. We then set $L_i = L_{i-1} + (L \cap \langle \mathbf{e}_1, \ldots, \mathbf{e}_{|\mathfrak{g}_{\mathsf{vk}}|} \rangle)$. (Intuitively, the right summand contains those linear constraints about $\mathfrak{g}_{\mathsf{vk}}$ that are implied by L.)

As before, note that step 2 can occur at most a finite number of times, because each time the dimension of L_i increases by at least 1 compared to L_{i-1} (if not, then L itself would have been a valid choice for L_{sim} in step 1). Further, step 1 does not require any GGM operations since it relies solely on Sim.

Dealing with random oracles. Before looking into blind signatures, let us see what happens when adding a random oracle (RO). In the GGM+ROM, verification is additionally allowed to make random oracle queries. We write them as $y \leftarrow \mathcal{H}(\mathfrak{h}, x)$, where $\mathfrak{h}$ is a vector of group elements, and x, y are both bit-strings.[8] To show an impossibility, we would need to find an attacker that is query-efficient in the number of queries to both the group oracle *and the RO*.

Note that equality-checks may depend on previous random oracle outputs. For example, a Schnorr signature has format $\sigma_i = (\mathfrak{g}_i, z_i)$ such that $\mathfrak{g}_i + c_i \cdot \mathfrak{g}_{\mathsf{vk}} - z_i \cdot \mathfrak{1} = \mathfrak{o}$ is fulfilled, where the public key consists of two group elements $\mathfrak{g}_{\mathsf{vk}} = (\mathfrak{1}, \mathfrak{g}_{\mathsf{vk}})$. A forgery would be trivial if it weren't for the fact that c_i is the output $\mathcal{H}(\mathfrak{g}_i, m_i)$ of the RO on input $\mathfrak{g}_i$ and the message.

We can still attempt to construct simulator Sim as before, but it gets more involved since now it will also need to simulate RO queries (otherwise, running Sim would not be free anymore in terms of queries!). In general, we simulate all RO outputs as uniformly random, while ensuring consistency between several queries to $\mathcal{H}$. The simulator will therefore keep track of a list $\mathcal{S}$ of previously-seen inputs. Its entries have the form (T, x, y) where T is a matrix that describes the group elements queried to the random oracle (similar to what we previously denoted by $\mathbf{v}^\mathsf{T}$ in (2), except that T contains multiple rows). More precisely, Sim would start out with $\mathcal{S} := \emptyset$, and replace RO calls in Verify by the following:

$$
\begin{array}{lcl}
\underline{\mathsf{Verify}((s, \mathfrak{g}_{\mathsf{vk}}), m, (t, \mathfrak{g}_\sigma))} & & \underline{\mathsf{Sim}((s, \mathfrak{g}_{\mathsf{vk}}), m, (t, L_{\mathsf{sim}}))} \\
\texttt{//some code here} & & \texttt{//some code here} \\
y \leftarrow \mathcal{H}(T \cdot \begin{pmatrix} \mathfrak{g}_{\mathsf{vk}} \\ \mathfrak{g}_\sigma \end{pmatrix}, x) & \rightsquigarrow & \textbf{if } (T', x, y') \in \mathcal{S} \text{ with } \mathrm{rowsp}(T - T') \subseteq L_{\mathsf{sim}} \\
 & & \textbf{then } y \leftarrow y' \\
 & & \textbf{else } y \leftarrow\!\!\$\ \{0,1\}^k; \mathcal{S} := \mathcal{S} \cup \{(T, x, y)\} \\
\texttt{//more code here} & & \texttt{//more code here}
\end{array}
\tag{3}
$$

[7] We are sweeping under the rug some technicalities: Note that the chosen $\mathfrak{g}_\sigma^*$ may accidentally satisfy an equality check $\mathbf{v}$ (see left-hand side of (2)), even though Sim simulated it as 0 due to $\mathbf{v} \notin L_{\mathsf{sim}}$. Then, σ^* is not necessarily a valid forgery. In order to mitigate this, we will actually choose $\mathfrak{g}_\sigma^* := A^+ \cdot B \cdot \mathfrak{g}_{\mathsf{vk}} + (A^+ \cdot A - I) \cdot \mathbf{z} \cdot \mathfrak{1}$ for some uniformly random $\mathbf{z} \leftarrow\!\!\$\ \mathbb{Z}_p^{|\mathfrak{g}_\sigma|}$. This injects sufficient randomness to ensure that if the resulting forgery σ^* still is not valid, then w.h.p., we can infer from this a new constraint $\mathbf{v} \in \langle \mathbf{e}_1, \ldots, \mathbf{e}_{|\mathfrak{g}_{\mathsf{vk}}|} \rangle$ that we may add to L_i, similar to case 2.

[8] For now we assume that the output of $\mathcal{H}$ is a bitstring, i.e., it does not contain group elements. We later describe how to lift this restriction.

Note that "rowsp$(T - T') \subseteq L_{\mathsf{sim}}$" can be thought of as checking whether the two group element vectors (described by T and T' w.r.t. a base consisting of $\mathfrak{g}_{\mathsf{vk}}$ and $\mathfrak{g}_\sigma$) are identical, *according to the simulation space* L_{sim}.

Note that Sim is not deterministic anymore, and therefore the attack from before would have to be changed s.t. step 1 attempts to find (t^*, L_{sim}) with say

$$\Pr[\mathsf{Sim}((s, \mathfrak{g}_{\mathsf{vk}}), m_i, (t^*, L_{\mathsf{sim}})) \text{ outputs } 1] \geq \frac{1}{2}\,, \tag{4}$$

i.e., the "simulated" success probability is sufficiently high.

Of course, for plain signatures, this strategy cannot successfully create forgeries for arbitrary schemes (since Schnorr signatures are provably secure in the GGM+ROM). In the attack from before, step 2 is what fails now due to the added RO: The space $L \cap \langle \mathbf{e}_1, \ldots, \mathbf{e}_{|\mathfrak{g}_{\mathsf{vk}}|} \rangle$ of "learned" linear constraints about vk will be empty, and therefore the dimension of L_i does not increase at all.

In the following, we will however see how to make such an attack work in the setting of blind signatures.

Blind signatures. It is first convenient to introduce some notation–whenever we talk about a round-optimal blind signature scheme, we think of the user first running an algorithm $\mathsf{User}_1(\mathsf{vk}, m)$ that takes as input the public verification key together with the message to be signed, and it outputs the message msgU meant for the signer, as well as some state stU to be kept locally by the user. The signer then runs $\mathsf{Sign}(\mathsf{sk}, \mathsf{msgU})$ (where sk is the signing key), and this produces a response msgS. Finally, the user runs $\mathsf{User}_2(\mathsf{msgS}, \mathsf{stU})$ to produce signature σ.

Blindness restricts the signer's random oracle queries. A first helpful observation that was already made by [41] is that random oracle queries that are made both by the signer as well as the verification algorithm may be used to link signing sessions and signatures, thus breaking blindness. The only case where such queries do not break blindness is when they are "irrelevant" in the sense that they are not specific to the message-signature pair, i.e., they are made during the verification of many message-signature pairs (say, upon signing a random message). However, such irrelevant queries are easy for us to learn by simply requesting several signatures on random messages and observing the random oracle queries made during verification, and adding those to $\mathcal{S}$ before running the simulator Sim. For the purpose of this technical overview, we therefore assume for most parts that the signer is guaranteed to never make RO queries that appear again later during Verify.

First attempt: Straight to the signature. Given that observation, one might ask whether the described attack strategy (i.e., given a message m, iterating through all possible choices of (t^*, L_{sim}), checking (4) for each of them and if true, construct $\mathfrak{g}_\sigma$ from $\mathfrak{g}_{\mathsf{vk}}$ and L_{sim}) is already sufficient in the blind signature setting. After all, the Schnorr scheme (which, when transformed into the blind signature setting would mean $\mathsf{msgU}_i := m_i$, and the signature $\sigma_i := \mathsf{msgS}_i := (\mathfrak{g}_i, z_i)$ is taken verbatim from the signer) that was a counterexample for the plain signature setting, would now contradict the assumption that the signer never makes the same RO query as the verifier.

Let us put aside for now the issue of *learning*, i.e., finding a way to ensure that in each step 2, the dimension of space L_i increases by at least 1 compared to L_{i-1}. Even if that is taken care of, it may be straight up impossible to find any (t^*, L_{sim}) for which (4) holds. The issue is that, in a blind signature protocol, some random oracle queries may already be made by the *user* during the issuance protocol such that they are not "fresh" during verification, thus introducing additional interdependencies between RO input-output pairs and parts of the signature such as t.

As a very simple counterexample, suppose that some blind signature scheme requires the user to make a query $y \leftarrow \mathcal{H}(\mathfrak{g}_{\mathsf{vk}}, m)$, and the signature consists only of the bit-string $\sigma := y$. Then, Verify checks whether $y = \mathcal{H}(\mathfrak{g}_{\mathsf{vk}}, m)$ indeed holds. Clearly, the simulator Sim, regardless of the choice of (y^*, L_{sim}), will be "incorrect" in the sense that it simulates the output y of $\mathcal{H}(\mathfrak{g}_{\mathsf{vk}}, m)$ as uniformly random, which is equal to y^* with no more than negligible probability.

The user's random oracle queries. Of course the forger can easily run $\mathsf{msgU}, \mathsf{stU} \leftarrow \mathsf{User}_1(m)$ on the target message m to obtain the queries that are made during User_1, and immediately add them to the set $\mathcal{S}$ in order to aid the simulator Sim. However, the same does not apply to User_2 (which would require as input a signer message msgS). Therefore, our goal will be the following: Instead of directly forging a signature $\sigma = (\mathfrak{g}_\sigma, t)$, we aim to "forge" a message $\mathsf{msgS} = (\mathfrak{g}_{\mathsf{msgS}}, t_{\mathsf{msgS}})$ sent from the signer to the user, for which the simulation of User_2+Verify has high success probability.

More precisely, instead of letting Sim be transformations (2) and (3) applied to Verify, we apply these two transformations to the following combined procedure (taking as input the user state stU in addition to vk, m, and msgS):

$$\sigma \leftarrow \mathsf{User}_2(\mathsf{stU}, \mathsf{msgS}) \; ; \quad \textbf{return } \mathsf{Verify}(\mathsf{vk}, m, \sigma)$$

We note that by doing so, group-equality checks within User_2 and Verify are no longer made using representations for a base consisting of $\mathfrak{g}_{\mathsf{vk}}, \mathfrak{g}_\sigma$, but instead for a base that consists of $\mathfrak{g}_{\mathsf{vk}}, \mathfrak{g}_{\mathsf{msgS}}$. Thus, L_{sim} must no longer be a subspace of $\mathbb{Z}_p^{|\mathfrak{g}_{\mathsf{vk}}|+|\mathfrak{g}_\sigma|}$, but instead a subspace of $\mathbb{Z}_p^{|\mathfrak{g}_{\mathsf{vk}}|+|\mathfrak{g}_{\mathsf{msgS}}|}$.

When the attacker finds some $(t^*_{\mathsf{msgS}}, L_{\mathsf{sim}})$ for which the probability in (4) is sufficiently large, it will be able to construct $\mathfrak{g}^*_{\mathsf{msgS}}$ from $\mathfrak{g}_{\mathsf{vk}}$ and L_{sim} as before. In order to construct the actual *signature* forgery, it remains to compute $\sigma^* \leftarrow \mathsf{User}_2(\mathsf{stU}, \mathsf{msgS})$ on $\mathsf{msgS} = (\mathfrak{g}^*_{\mathsf{msgS}}, t^*_{\mathsf{msgS}})$ (in the *real* world, not as a simulation!)

The signer's random oracle queries, revisited. RO queries made by User_1 and User_2 are now taken care of by our simulator. However, recall that this requires Sim to not only simulate Verify, but even User_2. This may cause new issues, since there is no guarantee that all queries made in User_2 (apart from those also made in User_1) are fresh. While we managed to exclude queries that are made by both Sign and Verify (by utilizing blindness), this does not apply to Sign and User_2.

To illustrate this issue, consider another contrived scheme: User_1 sends the message m to the signer, who replies with a Schnorr signature $\mathsf{msgS} = (\mathfrak{g}_i, z_i)$. Then, $\mathsf{User}_2(\mathsf{stU}, \mathsf{msgS})$ checks that msgS is indeed a valid Schnorr signature for m, and *if not, outputs* $\sigma = \bot$. Verify, on the other hand, always outputs 1 (unless $\sigma = \bot$). In this case, the attacker won't ever be able to come up with any $(t^*_{\mathsf{msgS}}, L_{\mathsf{sim}})$ for which the probability in (4) is more than negligible, because that requires knowing the dlog of vk's group elements.

This issue is resolvable due to the following intuition: since the RO query $y \leftarrow \mathcal{H}(\mathfrak{g}_i, m)$ is not made in Verify, it is not necessary to run User_2 using the *real* RO output y. Instead, we may simply "pretend" that y has some fixed value, e.g. $y = 0$, and L_{sim} may even depend on this chosen value.

More precisely, we modify the simulator Sim so that it additionally accepts a set of "fake" queries $\mathcal{S}_{\mathsf{fake}}$. Whenever User_2 would make a query in $\mathcal{S}_{\mathsf{fake}}$, then the corresponding fake output would be used by Sim instead of sampling a random one. On the other hand, when Verify would make some query, a normal simulation as before must be used. Then, the attack does not only iterate through all $(t^*_{\mathsf{msgS}}, L_{\mathsf{sim}})$ and compute the corresponding probability in (4), but it will iterate through all triples $(t^*_{\mathsf{msgS}}, L_{\mathsf{sim}}, \mathcal{S}_{\mathsf{fake}})$ (where $\mathcal{S}_{\mathsf{fake}}$ does not have to be consistent with the real RO) and compute that probability (using the modified Sim).[9]

Learning knowledge. Now that we know how to simulate for free given some already-learned knowledge L_i, we want to look at how the adversary obtains these L_i's while ensuring that its dimension indeed grows in each iteration. Previously, as part of step 2 in each iteration of the basic attack outline for generalized *plain* signature schemes, we simply ran $\mathsf{Verify}(\mathsf{vk}, m, \sigma)$, keeping track of a vector space L that contains all vectors $\mathbf{v}$ for which an equality-check returns 1. Naïvely generalizing this to the blind signature setting, we could attempt to now keep track of all equality-checks $\mathbf{v}$ (where

[9] We gloss over another detail here: In our actual attack, in order to avoid exceeding even polynomial space by having to iterate through all exponentially-sized sets $\mathcal{S}_{\mathsf{fake}}$, we also introduce "fake-relations" L_{fake} used for answering fake queries in $\mathcal{S}_{\mathsf{fake}}$ while simulating User_2. Then, $|\mathcal{S}_{\mathsf{fake}}|$ is bounded by the number of queries in Sign.

$\mathbf{v}$ now is a representation vector w.r.t. the basis that contains $\mathfrak{g}_{\mathsf{vk}}$ and $\mathfrak{g}_{\mathsf{msgS}}$ instead of $\mathfrak{g}_\sigma$) during both $\sigma \leftarrow \mathsf{User}_2(\mathsf{stU}, \mathsf{msgS})$ and $\mathsf{Verify}(\mathsf{vk}, m, \sigma)$.

Unfortunately, this is insufficient to ensure growing knowledge in each iteration in our interactive case. To see this, we consider yet another contrived scheme to produce Schnorr signatures: the public key contains group elements $\mathfrak{1}$ and $\mathfrak{g}_{\mathsf{vk}}$ s.t. $\mathfrak{g}_{\mathsf{vk}} = x \cdot \mathfrak{1}$. Then, User_1 sends an empty message to Sign, who always replies with the discrete log x together with Schnorr-commitment $\mathfrak{g}_i = r_i \cdot \mathfrak{1}$ and its dlog r_i. User_2 uses the dlogs x, r_i and the commitment $\mathfrak{g}_i$ in order to run $c_i \leftarrow \mathcal{H}(\mathfrak{g}_i, m)$ and compute a Schnorr signature $\sigma := (\mathfrak{g}_i, z_i)$ that is verified in Verify (for simplicity, we omit blinding the Schnorr commitment here, but the protocol can be made blind using standard Schnorr blinding techniques). Note that there are no RO queries made by both Sign and Verify, and therefore our attack needs to be able to handle such a scheme.

Learning more knowledge. While this scheme is obviously insecure, our generic attacker is oblivious to the relations $\mathfrak{g}_{\mathsf{vk}} = x \cdot \mathfrak{1}$, and $\mathfrak{g}_i = r_i \cdot \mathfrak{1}$ with the values x, r_i from the signer's message and will not learn them from merely observing a single run of User_2 and Verify. It would only see the following relation on the group elements:

$$\mathfrak{g}_i + c_i \cdot \mathfrak{g}_{\mathsf{vk}} - z_i \cdot \mathfrak{1} = \mathfrak{o} \ .$$

The space of "learned constraints" L about $\mathfrak{g}_{\mathsf{vk}}, \mathfrak{g}_{\mathsf{msgS}}$ merely consists of $L = \left\langle (-z_i, c_i, 1)^\mathsf{T} \right\rangle$. This does not imply any linear constraints on $\mathfrak{g}_{\mathsf{vk}}$ by itself, i.e., $L \cap \langle \mathbf{e}_1, \mathbf{e}_2 \rangle = \{\mathbf{0}\}$, and therefore our knowledge $L_i = L_{i-1} = \{\mathbf{0}\}$ stays trivial.

This means that we cannot find any triple $(t^*_{\mathsf{msgS}}, L_{\mathsf{sim}}, \mathcal{S}_{\mathsf{fake}})$ for which our simulator has high success probability, because Sim will need to output 1 for a large fraction of potential RO outputs $\mathcal{H}(\mathfrak{g}, m)$, whereas the bit-strings $t^*_{\mathsf{msgS}} = (x, r_i)$ must have been chosen *before* starting the simulator. We want to use this fact to allow our adversary to extract additional information from the signer's message. We employ a strategy of *re-randomizing* the random oracle for a single msgS many times, i.e., re-running User_2+Verify several times on the same user state stU_i and signer message msgS_i, but with different random oracle responses (thus, we learn all relations that are relevant for most ROs). In example above we would learn one additional equation when re-randomizing just once. We get

$$\mathfrak{g}_i + c_i \cdot \mathfrak{g}_{\mathsf{vk}} - z_i \cdot \mathfrak{1} = 0 \quad \text{and} \quad \mathfrak{g}_i + c'_i \cdot \mathfrak{g}_{\mathsf{vk}} - z'_i \cdot \mathfrak{1} = 0 \ ,$$

i.e., the space of "learned constraints" L about $\mathfrak{g}_{\mathsf{vk}}, \mathfrak{g}_{\mathsf{msgS}}$ now is w.h.p. two-dimensional: $L = \left\langle (-z_i, c_i, 1)^\mathsf{T}, (-z'_i, c'_i, 1)^\mathsf{T} \right\rangle$. Thus, the attack learns the non-trivial linear constraints $L \cap \langle \mathbf{e}_1, \mathbf{e}_2 \rangle = \left\langle (-x, 1, 0)^\mathsf{T} \right\rangle$ about the verification key (i.e., it learns the discrete log of $\mathfrak{g}_{\mathsf{vk}}$).

In the blind signature scheme just considered, there is only one random oracle query per signing session and it is made for the first time during User_2. In general, there may however be several random oracle queries during User_2, some of which may have already been made before by Sign or KeyGen. Even if these queries might not appear during verification later on, re-randomizing their responses might result in invalid or no signatures. Ideally we would therefore like to *only* re-randomize those queries that are made for the first time by User_2.

Suppose we are capable of performing re-randomization in this way, and we obtained many signatures $\sigma_1, \ldots, \sigma_\ell$ for the same message m_i (but different RO's, except for queries that were already fixed by KeyGen, User_1, and Sign). Combining all relations from all of these signatures, we can learn more linear constraints than with just one signature. If, for m_i and corresponding signer-message $\mathsf{msgS}_i = (t_{\mathsf{msgS}}, \mathfrak{g}_{\mathsf{msgS}})$, even after running a large (but polynomial) number of re-randomizations, the space L of implied constraints does not contain any new information, i.e., $L \cap \left\langle \mathbf{e}_1, \ldots, \mathbf{e}_{|\mathfrak{g}_{\mathsf{vk}}|} \right\rangle = L_{i-1}$, then simulation (as per (4)) should have had large success probability[10] on the triple $(t_{\mathsf{msgS}}, L, \mathcal{S}_{\mathsf{fake}})$, with $\mathcal{S}_{\mathsf{fake}}$ being chosen as the set of *real* RO queries made by Sign. Hence, we would have been able to construct a signature in step 1 by ourselves.

[10] The rough intuition for this is that the probability of Sim returning 1 can be seen to be taken over *re-randomizations* of the RO, and therefore is identical to a "real" re-randomization given the real msgS. However, formalizing this intuition requires a lot of care, which is done in the main body of this paper.

One technical difficulty comes from the fact that it is difficult for the attacker to determine whether a given query has been made before by KeyGen or Sign, and hence whether it should be re-randomized or not. This is the reason why our impossibility result is restricted to $q \leq O(\log \lambda)$ many queries during User_2 and Verify combined: the attack performs re-randomization by just guessing for every query whether it was made before during this signing session or not. This is successful only with probability 2^{-q}. We stress that the number of queries during KeyGen, User_1, and Sign is not affected by this.

Other technical challenges. Earlier on, we have made the simplifying assumption that the verifier never makes any RO query that has also been made by the signer (justified using the blindness property). Translating this intuition into a rigorous impossibility proof is possible (we refer the reader to the main body of this paper for the details) but complicates many things. For example, while "irrelevant" queries (that occur during the signing session for a large number of messages) are indeed easily learned by the attacker, this is "imperfect" in the sense that the attacker only learns a *superset* of those queries, and cannot easily tell which of them are indeed irrelevant and which ones are not. This affects our simulator Sim in a subtle way, because Sim treats *all* learned queries as fixed, even including some non-irrelevant ones. Thus, it is not a priori clear that Sim's success probability (4) is still sufficiently high despite knowing "too many" queries. To prove this, we need a technical lemma showing that, given a procedure that makes at most q RO queries and has probability p of returning 0, after fixing some arbitrary number of RO outputs and simulating all others in fresh way, the resulting procedure still has probability $\leq 2^q \cdot p$ of returning 0.

As another technicality, we note that when learning "irrelevant" queries, the attacker may not necessarily have representations of their input elements w.r.t. base $\mathfrak{g}_{\mathsf{vk}}$, but only w.r.t. $\mathfrak{g}_{\mathsf{vk}}$ plus elements from msgS_i (for many different i). Hence, the final attacker will in fact need to keep state of a large "pile" of group elements, denoted by $\mathfrak{g}$, which not only includes $\mathfrak{g}_{\mathsf{vk}}$ but also $\mathfrak{g}_{\mathsf{msgS}_i}$ for many i. Forgeries will need to be constructed from $\mathfrak{g}$ instead of just $\mathfrak{g}_{\mathsf{vk}}$.

ROs that output group elements. Finally, we note that our impossibility extends an RO that outputs group elements in addition to bitstrings: In transformation (3), we would additionally need to handle returning random group elements. An important observation is that it is very unlikely that a random group element is non-trivially related to other group elements, and therefore the simulator can simply return random representations. See Section 3.10 for details.

2 Preliminaries

2.1 Notation

We denote vectors $\mathbf{v}$ in bold face and matrices using capital letters. $\langle \mathbf{v}_1, \ldots, \mathbf{v}_n \rangle$ denotes the vector space of all linear combinations of $\mathbf{v}_1, \ldots, \mathbf{v}_n$. We denote by $\mathbb{Z}_p^\infty$ the set of vectors with entries in $\mathbb{Z}_p$ that have finitely many non-zero entries. We use $\mathbf{v}_i$ to denote the i-th entry of vector $\mathbf{v}$, and similarly M_i denotes the i-th row of a matrix M. We denote by $\mathbf{e}_i$ the ith unit vector in $\mathbb{Z}_p^\infty$. By default, all vectors are column vectors. Two vectors, arrays, or bitstrings can be concatenated using "$\|$". The bitwise XOR operation is denoted by "$\oplus$".

In pseudocode, we use ":=" for assignment, "$\leftarrow$" for assignment from a potentially randomized process, and "$\leftarrow\!\!\$$" for assignment from a uniform distribution specified by an explicit set on the right-hand side.

We use A^O to denote the execution of an algorithm A with oracle access to another algorithm O. The notation $A[x]$ is used when a value x is hard-coded into A, and $A^{[x]}$ denotes that A has read-and-write access to the variable x that may be modified both outside and inside of A. For any two oracles O_1, O_2, we will use the notation $O_1 \prec O_2$ to denote a "fallback", i.e., it denotes a new oracle O that, on input x, first runs $y_1 \leftarrow O_1(x)$ and returns y_1 if $y_1 \neq \bot$. Whenever $y_1 = \bot$, O additionally runs $y_2 \leftarrow O_2(x)$ and returns y_2 instead.

2.2 Generic Group Model

We will consider the same version of Maurer's Generic Group Model as used e.g. by Döttling et al. in [29], and what is denoted the "type safe" GGM in [56] (except that the latter only supports circuits instead of arbitrary algorithms). In order to rigorously prove our main result, we need a more formal version of those GGM definitions, which is given in this section.

In order to provide a clear distinction between group elements and other objects, we write group elements in fraktur, e.g. $\mathfrak{g}$, and a vector of group elements in bold fraktur, e.g. $\mathfrak{g}$.

GGM algorithms. Intuitively, a GGM algorithm $\overline{A}$ operates on *labels* in $\mathbb{N}$ which reference group elements in some group $\mathbb{G}$ that are unknown to $\overline{A}$. Each input/output of A and each input/output of an oracle call made by A will only contain such labels. In order to make such a GGM algorithm $\overline{A}$ compatible with other algorithms, we define a non-GGM *transformation* A that works on real group elements in $\mathbb{G}$ instead of labels:

Definition 2.1 (Generic Group Model). *Let $\mathbb{G} = \mathbb{G}(\lambda)$ be a family of groups s.t. $\mathbb{G}$ (written additively) has prime order $p = p(\lambda)$. Further, let $\mathfrak{1} = \mathfrak{1}(\lambda)$ denote an arbitrary generator of $\mathbb{G}$, and let $\mathfrak{o} = \mathfrak{o}(\lambda)$ be $\mathbb{G}$'s neutral element.*

Let $\mathcal{O}_1^\lambda, \ldots, \mathcal{O}_\ell^\lambda$ be oracles (we omit λ when clear from context), with input/output space $\mathbb{G}^* \times \{0,1\}^*$ (i.e., consisting of an arbitrary number of group elements and a bitstring; we call an element in $\mathbb{G}^* \times \{0,1\}^*$ an object*).*

Then, we say that $\overline{A}$ is a GGM algorithm compatible with $\mathcal{O}_1, \ldots, \mathcal{O}_\ell$ *if it has oracle access to $\mathcal{O}_{\mathsf{eq}}$, $\mathcal{O}_{\mathsf{grp}}$, and $\overline{\mathcal{O}}_1, \ldots, \overline{\mathcal{O}}_\ell$, where:*

- *$\overline{A}$ takes an input $(1^\lambda, 1^n, x)$ for $\lambda, n \in \lambda$ and $x \in \{0,1\}^*$ (where n denotes the number of input labels and x is a bitstring), and produces an output of the form $(i_1, \ldots, i_m, y) \in \mathbb{N}^* \times \{0,1\}^*$ (where $i_1, \ldots, i_m$ denote labels and y is a bitstring),*
- *the* group equality *oracle $\mathcal{O}_{\mathsf{eq}}(i,j)$, given two labels $i, j \in \mathbb{N}$, returns a value in $\{0, 1, \bot\}$,*
- *the* group operation *oracle $\mathcal{O}_{\mathsf{grp}}(i,j)$, given two labels $i, j \in \mathbb{N}$, returns either a label in $\mathbb{N}$ or $\bot$, and*
- *each oracle $\overline{\mathcal{O}}_k$, given an input of the form $(i_1, \ldots, i_r, x) \in \mathbb{N}^* \times \{0,1\}^*$ (where $i_1, \ldots, i_r$ denote labels and x is a bitstring), returns an output of the form $(i'_1, \ldots, i'_s, y) \in \mathbb{N}^* \times \{0,1\}^*$ or $\bot$ (where $i'_1, \ldots, i'_s$ denote labels and y is a bitstring).*

For any such GGM algorithm $\overline{A}$, we define a transformation that results in a (non-GGM) algorithm A that has access to oracles $\mathcal{O}_1, \ldots, \mathcal{O}_\ell$. This transformation is defined in Figure 1.

For blind signatures, we will actually require algorithms whose input/output consists of multiple objects, i.e., has the form $(\mathbb{G}^* \times \{0,1\}^*)^k$ for some $k \in \mathbb{N}$. For example, User_1 takes two objects (vk and message m), and produces two objects (msgU and stU). The above definition can easily extended to handle these cases.

2.3 Random Oracles

Definition 2.2 (Random Oracle for Groups). *A Random Oracle (RO) $\mathcal{H}$ with n^{H} input group elements and an output of bitlength r is an oracle that implements a function with input space $\mathbb{G}^{n^{\mathsf{H}}} \times \{0,1\}^*$ and output space $\{0,1\}^r$. Unless the function table of $\mathcal{H}$ has been fixed beforehand, using $\mathcal{H}$ inside a probability implicitly means that the function table is chosen* uniformly at random.

Remark 2.3. One could also define random oracles to have output space $\mathbb{G}^{m^{\mathsf{H}}} \times \{0,1\}^r$ instead of just $\{0,1\}^r$. For simplicity of our lower bound, we assume that the RO only outputs bitstrings, but we remark that it is possible to extend the lower bound to this more general setting (see Section 3.10).

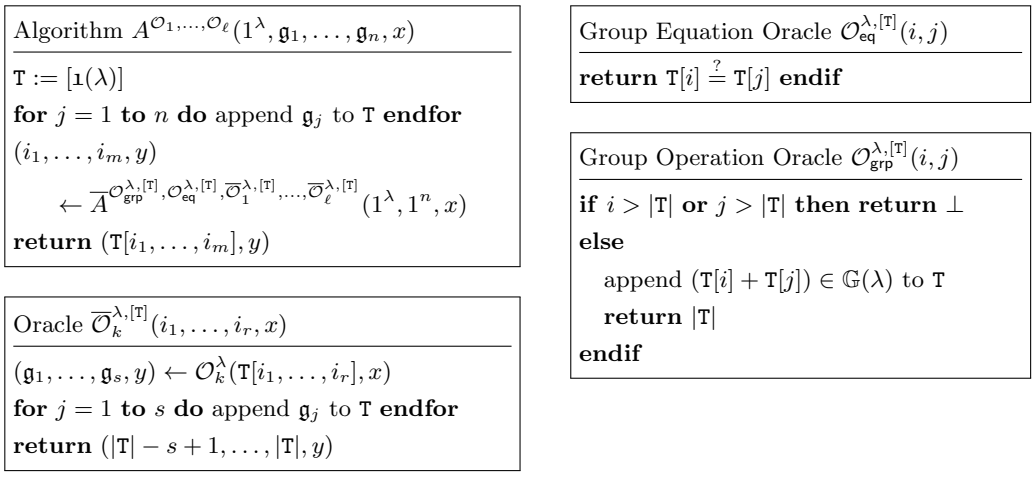

Fig. 1: Definition of the transformation A for some GGM algorithm $\overline{A}$ (top-left). We use $\mathtt{T}$ to denote an array of group elements from $\mathbb{G}$, initially containing only the generator $\mathfrak{1}$ of $\mathbb{G}$. $|\mathtt{T}|$ denotes the current length of $\mathtt{T}$ (initially 1), $\mathtt{T}[i] \in \mathbb{G} \cup \{\perp\}$ denotes the i-th element in the table (1-indexed) or $\perp$ if $i \notin \{1, \ldots, |\mathtt{T}|\}$, and $\mathtt{T}[i_1, \ldots, i_r]$ denotes the vector $(\mathtt{T}[i_1], \ldots, \mathtt{T}[i_r])$ of group elements. Inside A, the GGM algorithm $\overline{A}$ is run using the two group oracles $\mathcal{O}_{\mathsf{eq}}$ and $\mathcal{O}_{\mathsf{grp}}$ (right-hand side) as well as $\overline{\mathcal{O}}_k$ (bottom-left), all of which have (read and write) access to the table $\mathtt{T}$.

As an example, a GGM algorithm compatible with $\mathcal{H}$ would have the form $\overline{A}^{\mathcal{O}_{\mathsf{eq}}, \mathcal{O}_{\mathsf{grp}}, \overline{\mathcal{H}}}$, where $\overline{\mathcal{H}}$ has the interface $\mathbb{N}^{n^{\mathsf{H}}} \times \{0,1\}^* \to \{0,1\}^*$. Running the transformation $A^{\mathcal{H}}$ requires access to some RO $\mathcal{H}$ as defined in Definition 2.2 with interface $\mathbb{G}^{n^{\mathsf{H}}} \times \{0,1\}^* \to \{0,1\}^*$. Note that there are some subtleties involving GGM algorithms: Consider two GGM algorithms $\overline{A}$ and $\overline{B}$, where $\overline{A}$ is compatible with $\mathcal{H}$, and $\overline{B}$ is compatible with both A and $\mathcal{H}$. Then, we may run the transformed algorithm $B^{A^{\mathcal{H}}, \mathcal{H}}$. However, the transformations A and B encapsulate *separate* tables $\mathtt{T}$ when running $\overline{A}$ and $\overline{B}$, respectively. Thus, the labels used by $\overline{A}$ for invoking $\overline{\mathcal{H}}$ are entirely independent of the labels used by $\overline{B}$ for invoking $\overline{\mathcal{H}}$, but $\overline{A}$ and $\overline{B}$ would still receive the same outputs if the associated group elements are the same.

2.4 Blind Signatures

Definition 2.4 (Blind Signatures). *Let $\mathbb{G} = \mathbb{G}(\lambda)$ be a family of groups s.t. $\mathbb{G}$ has prime order $p = p(\lambda)$. Then, a* round-optimal blind signature scheme $\mathsf{BS} = (\mathsf{KeyGen}, \mathsf{User}_1, \mathsf{Sign}, \mathsf{User}_2, \mathsf{Verify})$ *in the ROM with message space $\mathcal{M} = \mathcal{M}(\lambda)$ consists of five ppt algorithms with access to an RO $\mathcal{H}$:*[11]

$\mathsf{KeyGen}^{\mathcal{H}}(1^\lambda)$ *The key generation algorithm outputs a keypair $(\mathsf{sk}, \mathsf{vk})$ where sk is a bitstring, and vk is of the form $(\mathbf{u}^{\mathsf{vk}}, s^{\mathsf{vk}}) \in \mathbb{G}^{n^{\mathsf{vk}}} \times \{0,1\}^*$.*

$\mathsf{User}_1^{\mathcal{H}}(\mathsf{vk}, m)$ *On input of a verification key vk and a message $m \in \mathcal{M}$, the first user algorithm outputs a user-message $\mathsf{msgU} = (\mathbf{u}^{\mathsf{msgU}}, s^{\mathsf{msgU}}) \in \mathbb{G}^{n^{\mathsf{msgU}}} \times \{0,1\}^*$ and an internal state $\mathsf{stU} = (\mathbf{u}^{\mathsf{st}}, s^{\mathsf{st}}) \in \mathbb{G}^{n^{\mathsf{st}}} \times \{0,1\}^*$.*

$\mathsf{Sign}^{\mathcal{H}}(\mathsf{sk}, \mathsf{msgU})$ *On input of a secret key sk and a user message msgU as above, the signer outputs a signer-message $\mathsf{msgS} = (\mathbf{u}^{\mathsf{msgS}}, s^{\mathsf{msgS}}) \in \mathbb{G}^{n^{\mathsf{msgS}}} \times \{0,1\}^*$.*

$\mathsf{User}_2^{\mathcal{H}}(\mathsf{stU}, \mathsf{msgS})$ *On input of an internal state stU and a signer-message msgS as above, the user outputs a signature $\sigma = (\mathbf{u}^{\mathsf{sig}}, s^{\mathsf{sig}}) \in \mathbb{G}^{n^{\mathsf{sig}}} \times \{0,1\}^*$.*

[11] Note that this definition is unrelated to the GGM. The algorithms could avoid outputting group elements and instead pass around bitstrings only. However, without group elements, it is trivial to design a computationally unbounded attack.

$\mathsf{Verify}^{\mathcal{H}}(\mathsf{vk}, m, \sigma)$ *On input of a verification key* vk*, a message* m *and a signature* σ *as above, the verification algorithm* deterministically *outputs a bit* b*, where* $b = 1$ *indicates that the signature is valid and* $b = 0$ *indicates that it is not.*

In the above, n^{vk}*,* n^{st}*,* n^{msgU}*,* n^{msgS}*, and* n^{sig} *(denoting the number of group elements in their respective objects) are all fixed polynomials in* λ*. For any algorithm* A *from the above, we use* $q_A = q_A(\lambda)$ *to denote an upper bound on the number of calls to* $\mathcal{H}$ *made by* A*.*

When writing $\sigma \leftarrow \langle \mathsf{User}^{\mathcal{H}}(\mathsf{vk}, m), \mathsf{Sign}^{\mathcal{H}}(\mathsf{sk})\rangle$*, we mean the sequential execution of* $(\mathsf{msgU}, \mathsf{stU}) \leftarrow \mathsf{User}_1{}^{\mathcal{H}}(\mathsf{vk}, m)$*,* $\mathsf{msgS} \leftarrow \mathsf{Sign}^{\mathcal{H}}(\mathsf{sk}, \mathsf{msgU})$*, and* $\sigma \leftarrow \mathsf{User}_2{}^{\mathcal{H}}(\mathsf{stU}, \mathsf{msgS})$*.*

In the following, we will also just use the term "blind signature scheme" to refer to a round-optimal blind signature scheme in the ROM. We require the following properties from a blind signature scheme:

Definition 2.5 (Correctness). *A blind signature scheme* BS *is* correct *if*

$$\Pr\left[\begin{matrix}(\mathsf{sk},\mathsf{vk}) \leftarrow \mathsf{KeyGen}^{\mathcal{H}}(1^\lambda) \\ \langle \mathsf{User}^{\mathcal{H}}(\mathsf{vk}, m), \mathsf{Sign}^{\mathcal{H}}(\mathsf{sk})\rangle\end{matrix} : \mathsf{Verify}(\mathsf{vk}, m, \sigma) = 1\right] \geq 1 - \mathrm{negl}(\lambda)\ .$$

Definition 2.6 (Blindness). *The blindness game* $\mathbf{blind}_{\mathsf{BS},\mathcal{S}}$ *for a blind signature scheme* BS *and an adversary* $\mathcal{S}$ *is defined as follows: First, sample a bit* $b \leftarrow\!\!\$\ \{0,1\}$*. Then, with* $\mathcal{H}$ *being an RO and the remaining oracles defined in the following, output* $b' \leftarrow \mathcal{S}^{\mathcal{H},\mathcal{O}_{\mathsf{init}},\mathcal{O}_{\mathsf{User}_1,0},\mathcal{O}_{\mathsf{User}_1,1},\mathcal{O}_{\mathsf{User}_2,0},\mathcal{O}_{\mathsf{User}_2,1}}(1^\lambda)$*.*

$\mathcal{O}_{\mathsf{init}}(\mathsf{vk}, m_0, m_1)$**.** *May be called at most once. Stores* vk, m_0, m_1*.*

$\mathcal{O}_{\mathsf{User}_1,c}$ **for** $c \in \{0,1\}$**.** *Each of these be called at most once, and only if* $\mathcal{O}_{\mathsf{init}}$ *was called before. Runs* $(\mathsf{stU}_c, \mathsf{msgU}_c) \leftarrow \mathsf{User}_1^{\mathcal{H}}(\mathsf{vk}, m_{b\oplus c})$*, and returns* msgU_c*.*

$\mathcal{O}_{\mathsf{User}_2,c}(\mathsf{msgS}_c)$ **for** $c \in \{0,1\}$**.** *Each of these may be called at most once, and only if* $\mathcal{O}_{\mathsf{User}_1,c}$ *was called before. Store* $\sigma_{b\oplus c} \leftarrow \mathsf{User}_2^{\mathcal{H}}(\mathsf{stU}_c, \mathsf{msgS}_c)$*. If* $\mathcal{O}_{\mathsf{User}_2,1\oplus c}$ *was called before and* $\mathsf{Verify}^{\mathcal{H}}(\mathsf{vk}, m_c, \sigma_c) = 1$ *for* $c = 0, 1$*, return* (m_0, σ_0) *and* (m_1, σ_1)*. Otherwise, return* $\perp$*.*

We define the advantage $\mathrm{adv}^{\mathbf{blind}}_{\mathsf{BS},\mathcal{S}} := 2 \cdot \left|\Pr[\mathbf{blind}_{\mathsf{BS},\mathcal{S}} : b = b'] - \frac{1}{2}\right|$*.*

Definition 2.7 (One-More Unforgeability (OMUF)). *The one-more unforgeability game* $\mathbf{omuf}_{\mathsf{BS},\mathcal{U}}$ *for a blind signature scheme* BS *and an adversary* $\mathcal{U}$ *is defined as follows, where* $\mathcal{H}$ *is an RO.*

1. *Sample a keypair* $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}^{\mathcal{H}}(1^\lambda)$*.*
2. *Invoke* $(m_i, \sigma_i)_{i\in[\ell+1]} \leftarrow \mathcal{U}^{\mathcal{H},\mathcal{O}_{\mathsf{Sign}}}(1^\lambda, \mathsf{vk})$*, where* ℓ *is the number of calls to* $\mathcal{O}_{\mathsf{Sign}}$*, and* $\mathcal{O}_{\mathsf{Sign}}(\cdot)$ *is defined as* $\mathsf{Sign}^{\mathcal{H}}(\mathsf{sk}, \cdot)$*.*
3. *Output* 1 *if for all* $i \neq j \in [\ell+1]$ *it holds that* $m_i \neq m_j$ *and for all* $i \in [\ell+1]$*,* $\mathsf{Verify}(\mathsf{vk}, m_i, \sigma_i) = 1$*.*

We define the advantage $\mathrm{adv}^{\mathbf{omuf}}_{\mathsf{BS},\mathcal{U}} := \Pr[\mathbf{omuf}_{\mathsf{BS},\mathcal{U}} = 1]$*.*

3 Impossibility Result

We now state our main theorem that we will prove in this section.

Theorem 3.1. *There does not exist a round-optimal blind signature scheme* BS *in the ROM for which all of the following requirements are satisfied:*

- BS *fulfills correctness.*
- *For any* ppt *GGM*[12] *blindness adversary* $\mathcal{S}$*, the advantage* $\mathrm{adv}^{\mathbf{blind}}_{\mathsf{BS},\mathcal{S}}$ *is negligible in* λ*.*

[12] Normally we allow GGM algorithms to be computationally unbounded, but note that the lower bound becomes stronger by relaxing this to ppt.

- *For any* computationally unbounded *GGM OMUF adversary* $\overline{\mathcal{U}}$ *that makes at most a polynomial number of calls to its oracles* $\mathcal{O}_{\mathsf{grp}}$, $\overline{\mathcal{H}}$ *and* $\overline{\mathcal{O}_{\mathsf{Sign}}}$, *the advantage* $\mathrm{adv}^{\mathbf{omuf}}_{\mathsf{BS},\mathcal{U}}$ *is negligible in* λ.
- *The algorithms* User_1, User_2, *and* Verify *are transformations of GGM algorithms* $\overline{\mathsf{User}_1}$, $\overline{\mathsf{User}_2}$, *and* $\overline{\mathsf{Verify}}$, *respectively.*
- *The message space* $\mathcal{M}$ *has size super-polynomial in* λ.
- *The number of RO queries* $q := q_{\mathsf{User}_2} + q_{\mathsf{Verify}}$ *during* User_2 *and* Verify *is bounded by* $q \leq O(\log \lambda)$.

We start by proving a property that follows from blindness (Section 3.1), and then we use this for constructing a successful OMUF adversary (Section 3.2) that makes a polynomial number of oracle (group, RO, and signature) queries.

3.1 Blindness restricts relevant queries during verification

We define *irrelevant* hash queries which are intuitively those hash queries which are made during the verification of many potential signatures:

Definition 3.2. *For fixed keypair* $(\mathsf{sk},\mathsf{vk})$ *and RO* $\mathcal{H}$, *we say that a query* $Q \in \mathbb{G}^{n^{\mathsf{H}}} \times \{0,1\}^*$ *is* α*-irrelevant, if*

$$\Pr\left[\begin{array}{l} m \xleftarrow{\$} \mathcal{M} \\ \sigma \xleftarrow{\$} \langle \mathsf{User}^{\mathcal{H}}(\mathsf{vk},m), \mathsf{Sign}^{\mathcal{H}}(\mathsf{sk})\rangle \end{array} : Q \text{ is made during } \mathsf{Verify}^{\mathcal{H}}(\mathsf{vk},m,\sigma)\right] \geq \alpha\ .$$

We use $\mathcal{IRR}^{\mathsf{vk},\mathsf{sk}}_{\alpha,\mathcal{H}}$ *to denote the set of all* α-irrelevant *hash queries w.r.t. keypair* $(\mathsf{sk},\mathsf{vk})$ *and RO* $\mathcal{H}$. *Note that we will typically treat* $\mathcal{IRR}^{\mathsf{vk},\mathsf{sk}}_{\alpha,\mathcal{H}}$ *as a random variable (when* $(\mathsf{sk},\mathsf{vk})$ *and* $\mathcal{H}$ *are also random variables).*

We first prove a helpful property of irrelevant hash queries:

Lemma 3.3. *For any* $(\mathsf{vk},\mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$ *we have* $\left|\mathcal{IRR}^{\mathsf{vk},\mathsf{sk}}_{\alpha,\mathcal{H}}\right| \leq \alpha^{-1} \cdot q_{\mathsf{Verify}}$.

Proof. Any query $Q \in \mathcal{IRR}^{\mathsf{vk},\mathsf{sk}}_{\alpha,\mathcal{H}}$ needs to happen during verification of an α-fraction of valid message signature pairs. As each verification can make at most q_{Verify} queries, there can be at most $q_{\mathsf{Verify}} \cdot \alpha^{-1}$ distinct irrelevant queries. □

Now we present the main result regarding blindness.

Lemma 3.4. *If for a blind signature scheme* BS *it holds that*

$$\Pr\left[\begin{array}{l} (\mathsf{vk},\mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}^{\mathcal{H}}(1^\lambda) \\ m \xleftarrow{\$} \mathcal{M} \\ \sigma \leftarrow \langle \mathsf{User}^{\mathcal{H}}(\mathsf{vk},m), \mathsf{Sign}^{\mathcal{H}}(\mathsf{sk})\rangle \end{array} : \mathcal{Q}^{\mathsf{Verify}} \cap \mathcal{Q}^{\mathsf{Sign}} \setminus \mathcal{IRR}^{\mathsf{vk},\mathsf{sk}}_{\alpha,\mathcal{H}} \neq \emptyset\right] \geq \beta \tag{5}$$

for some $\alpha = \frac{1}{\mathrm{poly}(\lambda)}$ *and* β *(where* $\mathcal{Q}^{\mathsf{Verify}}$ *is the set of RO queries that the deterministic verifier* $\mathsf{Verify}^{\mathcal{H}}(\mathsf{vk},m,\sigma)$ *would make,* $\mathcal{Q}^{\mathsf{Sign}}$ *is the set of RO queries that the execution of* $\mathsf{Sign}^{\mathcal{H}}(\mathsf{sk},\mathsf{msgU})$ *made, and* $\mathcal{H}$ *is uniformly random), then there is a polynomial-time GGM blindness adversary* $\overline{\mathcal{S}}$ *that has advantage*

$$\mathrm{adv}^{\mathbf{blind}}_{\mathsf{BS},\mathcal{S}} \geq \beta - \alpha \cdot q_{\mathsf{Verify}} \cdot \frac{\log_2(\sqrt{\alpha}) - \lambda - \log_2 q_{\mathsf{Verify}}}{\log_2(1-\sqrt{\alpha})} - \sqrt{\alpha} \cdot q_{\mathsf{Sign}} - 2^{-\lambda}\ .$$

Remark 3.5. As a consequence, the blind signature scheme cannot fulfill blindness against all ppt GGM adversaries if inequality (5) holds for values $\beta = \frac{1}{\mathrm{poly}(\lambda)}$ and $\alpha = \left(\frac{\beta}{2q_{\mathsf{Verify}}(\lambda+q_{\mathsf{Sign}}+\log q_{\mathsf{Verify}})}\right)^2$ (with some arbitrary polynomial $\mathrm{poly}(\lambda)$).

Proof of Lemma 3.4. We describe a ppt strategy $\mathcal{A}$ for breaking blindness in the case that there is message m for which the signer makes a relevant hash query. (It is easy to see that $\mathcal{A}$ can be written as the transformation of some GGM algorithm $\overline{\mathcal{A}}$.)

First, the signer attempts to construct a set $\mathcal{C}$ that contains as many *irrelevant* hash queries as possible for $\sqrt{\alpha}$ which is $> \alpha$. The reason why we collect irrelevant queries w.r.t. $\sqrt{\alpha}$ instead of α is that this collection will yield many false positives. To form $\mathcal{C}$, the signer first samples a key pair $(\mathsf{vk},\mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$. It then samples $m_1,\ldots,m_g \xleftarrow{\$} \mathcal{M}$ for $g = \frac{\log_2(\sqrt{\alpha})-\lambda-\log_2 q_{\mathsf{Verify}}}{\log_2(1-\sqrt{\alpha})}$. For each m_i it simulates the signing protocol by taking the role of both the signer and the user to generate signatures σ_i. It makes a list of candidate irrelevant queries $\mathcal{C}$ by adding all queries that are made during $\mathsf{Verify}(\mathsf{vk},m_i,\sigma_i)$ for some i to $\mathcal{C}$.

By a union bound and Lemma 3.3, it holds that

$$\Pr[\mathcal{IRR}^{\mathsf{vk},\mathsf{sk}}_{\sqrt{\alpha},\mathcal{H}} \not\subseteq \mathcal{C}] \leq \frac{1}{\sqrt{\alpha}} \cdot q_{\mathsf{Verify}} \cdot (1-\sqrt{\alpha})^g$$

and plugging in the above value for g:

$$\begin{aligned}
&\frac{1}{\sqrt{\alpha}} \cdot q_{\mathsf{Verify}} \cdot (1-\sqrt{\alpha})^g \\
=&\frac{1}{\sqrt{\alpha}} \cdot q_{\mathsf{Verify}} \cdot (1-\sqrt{\alpha})^{\frac{\log_2(\sqrt{\alpha})-\lambda-\log_2 q_{\mathsf{Verify}}}{\log_2(1-\sqrt{\alpha})}} \\
=&\frac{1}{\sqrt{\alpha}} \cdot q_{\mathsf{Verify}} \cdot 2^{\log_2(\sqrt{\alpha})-\lambda-\log_2 q_{\mathsf{Verify}}} \\
=&2^{-\lambda}
\end{aligned}$$

The signer then picks two new random messages m_0', m_1' as the challenge and calls $\mathcal{O}_{\mathsf{init}}(\mathsf{vk},m_0',m_1')$. It honestly runs the signing protocol: first call $\mathsf{msgU}_c \leftarrow \mathcal{O}_{\mathsf{User}_1,c}()$, then compute $\mathsf{msgS}_c \leftarrow \mathsf{Sign}(\mathsf{sk},\mathsf{msgU}_c)$, and finally call $\mathcal{O}_{\mathsf{User}_2,c}(\mathsf{msgS}_c)$ (for $c \in \{0,1\}$). The latter results in the two signatures σ_0' and σ_1'. While running $\mathsf{Sign}(\mathsf{sk},\mathsf{msgU}_c)$, the signer keeps track of the hash queries it makes in the set $\mathcal{Q}_c^{\mathsf{Sign}}$. After obtaining the two signature pairs (m_0',σ_0') and (m_1',σ_1'), the signer runs the verification algorithm on both of them, keeping track of the hash queries that $\mathsf{Verify}(\mathsf{vk},m_c',\sigma_c')$ made in $\mathcal{Q}_c^{\mathsf{Verify}}$.

It now checks whether $\mathcal{Q}_0^{\mathsf{Verify}} \cap \mathcal{Q}_0^{\mathsf{Sign}} \setminus \mathcal{C} = \emptyset$ and if yes it returns $b'=1$, otherwise $b'=0$.

We compute the probability of success:

$$\begin{aligned}
&\mathrm{adv}^{\mathbf{blind}}_{\mathsf{BS}}(\mathcal{A}) \\
&= 2\left|\Pr[b=b'] - \frac{1}{2}\right| \geq 2\Pr[b=b'] - 1 \\
&= \Pr[b'=0|b=0] - \Pr[b'=0|b=1] && (6)\\
&= \Pr\left[\mathcal{Q}_0^{\mathsf{Verify}} \cap \mathcal{Q}_0^{\mathsf{Sign}} \setminus \mathcal{C} \neq \emptyset \middle| b=0\right] - \Pr\left[\mathcal{Q}_0^{\mathsf{Verify}} \cap \mathcal{Q}_0^{\mathsf{Sign}} \setminus \mathcal{C} \neq \emptyset \middle| b=1\right] && (7)\\
&\geq \Pr\left[\mathcal{Q}_0^{\mathsf{Verify}} \cap \mathcal{Q}_0^{\mathsf{Sign}} \setminus \mathcal{IRR}^{\mathsf{vk},\mathsf{sk}}_{\alpha,\mathcal{H}} \neq \emptyset \wedge \begin{array}{c}\mathcal{C} \cap \mathcal{Q}_0^{\mathsf{Verify}} \cap \mathcal{Q}_0^{\mathsf{Sign}} \\ \setminus \mathcal{IRR}^{\mathsf{vk},\mathsf{sk}}_{\alpha,\mathcal{H}} = \emptyset\end{array} \middle| b=0\right] \\
&\quad - \Pr\left[\mathcal{Q}_0^{\mathsf{Verify}} \cap \mathcal{Q}_0^{\mathsf{Sign}} \setminus \mathcal{C} \neq \emptyset \middle| b=1\right] && (8)\\
&\geq \beta - \alpha \cdot g \cdot q_{\mathsf{Verify}} - \Pr\left[\mathcal{Q}_0^{\mathsf{Verify}} \cap \mathcal{Q}_0^{\mathsf{Sign}} \setminus \mathcal{C} \neq \emptyset \middle| b=1\right] && (9)
\end{aligned}$$

$$
\begin{aligned}
&= \beta - \alpha \cdot g \cdot q_{\mathsf{Verify}} - \Pr\left[\mathcal{Q}_0^{\mathsf{Verify}} \cap \mathcal{Q}_0^{\mathsf{Sign}} \setminus \mathcal{C} \neq \emptyset \wedge \mathcal{IRR}^{\mathsf{vk},\mathsf{sk}}_{\sqrt{\alpha},\mathcal{H}} \subseteq \mathcal{C} \middle| b = 1\right] \\
&\qquad\qquad - \Pr\left[\mathcal{Q}_0^{\mathsf{Verify}} \cap \mathcal{Q}_0^{\mathsf{Sign}} \setminus \mathcal{C} \neq \emptyset \wedge \mathcal{IRR}^{\mathsf{vk},\mathsf{sk}}_{\sqrt{\alpha},\mathcal{H}} \not\subseteq \mathcal{C} \middle| b = 1\right] \\
&\geq \beta - \alpha \cdot g \cdot q_{\mathsf{Verify}} - \sqrt{\alpha} \cdot q_{\mathsf{Sign}} - \Pr\left[\mathcal{IRR}^{\mathsf{vk},\mathsf{sk}}_{\sqrt{\alpha},\mathcal{H}} \not\subseteq \mathcal{C}\right] \qquad (10) \\
&\geq \beta - \alpha \cdot q_{\mathsf{Verify}} \cdot \frac{\log_2(\sqrt{\alpha}) - \lambda - \log_2 q_{\mathsf{Verify}}}{\log_2(1-\sqrt{\alpha})} - \sqrt{\alpha} \cdot q_{\mathsf{Sign}} - 2^{-\lambda}
\end{aligned}
$$

where in (6) we reformulated the advantage for the distinguishing game, in (7) we replaced the b' with the set intersections that the adversary does, in (8) we used a union bound over the α-irrelevant queries not contained in $\mathcal{C}$, in (9) we used β from the lemma statement as well as a union bound (over all $g \cdot q_{\mathsf{Verify}}$ elements in $\mathcal{C}$) showing that the probability of the event $\mathcal{C} \cap \mathcal{Q}_0^{\mathsf{Verify}} \setminus \mathcal{IRR}^{\mathsf{vk},\mathsf{sk}}_{\sqrt{\alpha},\mathcal{H}} \neq \emptyset$ is at most $\alpha \cdot g \cdot q_{\mathsf{Verify}}$, in (10) we bounded the probability of the event $\mathcal{Q}_0^{\mathsf{Verify}} \cap \mathcal{Q}_0^{\mathsf{Sign}} \setminus \mathcal{IRR}^{\mathsf{vk},\mathsf{sk}}_{\sqrt{\alpha},\mathcal{H}} \neq \emptyset$ conditioned on $b = 1$, in which case the sets $\mathcal{Q}_0^{\mathsf{Sign}}$ and $\mathcal{Q}_0^{\mathsf{Verify}}$ are generated independently, which (by definition of an $\sqrt{\alpha}$-irrelevant query) means that for any fixed query in $\mathcal{Q}_0^{\mathsf{Sign}}$, the probability of also occuring in $\mathcal{Q}_0^{\mathsf{Verify}}$ is $\leq \sqrt{\alpha}$, and in the final inequality we used our previously shown bound on the probability of $\mathcal{IRR}^{\mathsf{vk},\mathsf{sk}}_{\sqrt{\alpha},\mathcal{H}} \not\subseteq \mathcal{C}$. □

3.2 Our OMUF Adversary

We will now describe our OMUF adversray. In order to minimize clutter, we directly define adversary $\mathcal{A}$ instead of the GGM adversary $\overline{\mathcal{A}}$, but it is clear that all steps of $\mathcal{A}$ cleanly correspond to GGM operations. For example, whenever we use fraktur, e.g. $\mathfrak{g}$, then the GGM adversary $\overline{\mathcal{A}}$ would store the group element $\mathfrak{g}$ in the form of a *label*. Whenever we do not use fraktur for some variable, then its value is accessible by $\overline{\mathcal{A}}$ as it is. We also write $\mathcal{H}$ instead of $\overline{\mathcal{H}}$ and $\mathcal{O}_{\mathsf{Sign}}$ instead of $\overline{\mathcal{O}_{\mathsf{Sign}},}$. We omit the explicit use of group oracles $\mathcal{O}_{\mathsf{eq}}$ and $\mathcal{O}_{\mathsf{grp}}$ by writing $\mathfrak{g} + \mathfrak{h}$ instead of the more cumbersome $\mathcal{O}_{\mathsf{grp}}(\mathfrak{g}, \mathfrak{h})$, and $\mathfrak{g} \stackrel{?}{=} \mathfrak{h}$ instead of $\mathcal{O}_{\mathsf{eq}}(\mathfrak{g}, \mathfrak{h})$.

The pile of group elements $\mathfrak{g}$. Forging will require us to construct a signature $\sigma = (\mathbf{u}^{\mathsf{sig}}, s^{\mathsf{sig}})$, with $\mathbf{u}^{\mathsf{sig}}$ depending on previously seen group elements. Hence, our adversary maintains a vector $\mathfrak{g} = \left(\mathfrak{g}[1], \mathfrak{g}[2] \ldots\right)^{\mathsf{T}} \in \mathbb{G}^\infty$ of group elements obtained so far "from the outside": Initially, after receiving $\mathsf{vk} = (\mathbf{u}^{\mathsf{vk}}, s^{\mathsf{vk}})$, we set $\mathfrak{g}[1] := \mathfrak{1}$ and $\mathfrak{g}[2..n^{\mathsf{vk}}+1] := \mathbf{u}^{\mathsf{vk}}$,[13] and whenever we make a query to the signing oracle to receive the message $\mathsf{msgS} = (\mathbf{u}^{\mathsf{sig}}, s^{\mathsf{sig}})$, we append $\mathbf{u}^{\mathsf{sig}}$ to $\mathfrak{g}$. At any time, $n^{\mathfrak{g}}$ denotes the number of slots that are "in use" (initially $n^{\mathfrak{g}} := n^{\mathsf{vk}} + 1$), i.e., only the first $n^{\mathfrak{g}}$ entries may be non-$\mathfrak{o}$.

Group element representations. Whenever $\mathcal{A}$ has a label to a group element $\mathfrak{h}$, then it also knows (or it is always able to recover by studying previously made group operations) a "representation" $\mathbf{t} \in \mathbb{Z}_p^\infty$ of $\mathfrak{h}$ in terms of $\mathfrak{g}$, i.e., a vector $\mathbf{t}$ that fulfills $\mathfrak{h} := \mathbf{t}^{\mathsf{T}}\mathfrak{g}$. (Similarly, a matrix $T \in \mathbb{Z}_p^{k \times \infty}$ may be used to describe the vector of group elements $\mathfrak{h} = T \cdot \mathfrak{g}$.)

Now, our OMUF adversary $\mathcal{A}$ uses $T^{\mathsf{vk}} = \left(\mathbf{e}_2 \ldots \mathbf{e}_{n^{\mathsf{vk}}+1}\right)^{\mathsf{T}}$ to refer to the group elements $\mathbf{u}^{\mathsf{vk}}$, because we always have $\mathbf{u}^{\mathsf{vk}} = T^{\mathsf{vk}} \cdot \mathfrak{g}$. We will use the notation $\widetilde{\mathsf{vk}} = (T^{\mathsf{vk}}, s^{\mathsf{vk}})$, to denote such an object consisting of a representation matrix together with a bitstring, as a counterpart to $\mathsf{vk} = (\mathbf{u}^{\mathsf{vk}}, s^{\mathsf{vk}})$ that consists of group elements and a bitstring. The same can be done for msgU, msgS and σ.

Since $\mathcal{A}$ will have to run the blind signature algorithms User_1, User_2, and Verify as subroutines and we still need to know explicit representations of all group elements returned by these three algorithms, we first transform their GGM counterparts $\overline{\mathsf{User}_1}$, $\overline{\mathsf{User}_2}$, and $\overline{\mathsf{Verify}}$ into new algorithms $\widetilde{\mathsf{User}_1}$, $\widetilde{\mathsf{User}_2}$, and $\widetilde{\mathsf{Verify}}$ that make group element representations explicit, see Figure 2.

[13] In the overview, we implicitly assumed that $\mathfrak{1}$ is always contained in vk, but for the purpose of this formal description, we explicitly add $\mathfrak{1}$ to $\mathfrak{g}$.

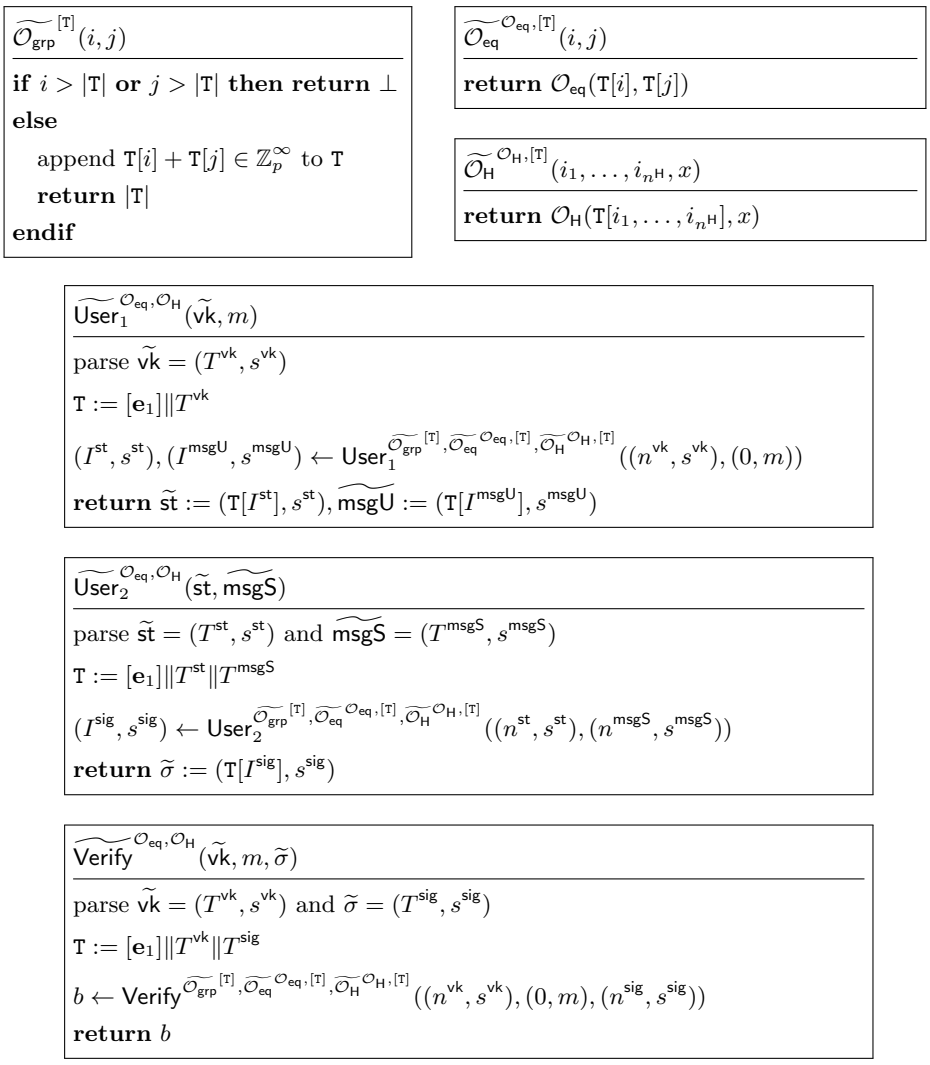

Fig. 2: Modified blind signature "GGM-like" algorithms $\widetilde{\mathsf{User}}_1$, $\widetilde{\mathsf{User}}_2$, and $\widetilde{\mathsf{Verify}}$. Instead of operating on labels the way a GGM algorithm would, $\widetilde{\mathsf{User}}_1$ operates on vectors in $\mathbb{Z}_p^\infty$, i.e., it queries $\mathcal{O}_{\mathsf{eq}}$ on a pair $(\mathbf{t}_1, \mathbf{t}_2) \in \mathbb{Z}_p^\infty$, its input is a pair $(\widetilde{\mathsf{vk}}, m)$ with $\widetilde{\mathsf{vk}} = (T^{\mathsf{vk}}, s^{\mathsf{vk}})$, $T^{\mathsf{vk}} \in \mathbb{Z}_p^{k \times \infty}$, etc. Analogous changes apply to $\widetilde{\mathsf{User}}_2$ and $\widetilde{\mathsf{Verify}}$. Note that there is no $\mathcal{O}_{\mathsf{grp}}$ needed when running any of these modified algorithms, because they can add two vectors on its own. The table $\mathtt{T}$ here contains representation vectors (from $\mathbb{Z}_p^\infty$) and should not be confused with the table from Figure 1 that maps labels to group elements.

Adversary overview. The adversary's outline is as follows.

Oracles $\mathcal{H}$, $\mathcal{O}_{\mathsf{Sign}}$
Input: $\mathsf{vk} = (\mathbf{u}^{\mathsf{vk}}, s^{\mathsf{vk}})$
Step 1: Initialization.

- We initialize $\mathfrak{g} := \left(\mathfrak{1}, \mathfrak{u}_1^{\mathsf{vk}} \ldots \mathfrak{u}_{n^{\mathsf{vk}}}^{\mathsf{vk}}, \mathfrak{o} \ldots\right)^{\mathsf{T}}$ to contain the generator $\mathfrak{1}$ as well as all vk-elements. Let $n^{\mathfrak{g}} := n^{\mathsf{vk}} + 1$ be the current number of "in use" elements.
- Define $\widetilde{\mathsf{vk}} := (T^{\mathsf{vk}}, s^{\mathsf{vk}})$, where $T^{\mathsf{vk}} := \left(\mathbf{e}_2 \ldots \mathbf{e}_{1+n^{\mathsf{vk}}}\right)^{\mathsf{T}}$ is the matrix that describes vk's group elements on basis $\mathfrak{g}$.
- Let $L := \{\mathbf{0}\} \subseteq \mathbb{Z}_p^\infty$ be an (initially trivial) vector space containing known linear relations about $\mathfrak{g}$. Let $\mathcal{S} = \emptyset$ be a set of known RO query input-output pairs.

Step 2: Learning Phase. See Figure 6.
Step 3: Forging Phase. See Figure 7.

In addition to the group elements $\mathfrak{g}$, the adversary maintains (1) a vector space $L \subseteq \mathbb{Z}_p^\infty$ containing linear relations about $\mathfrak{g}$, and (2) a set $\mathcal{S}$ of RO input-output pairs. In step 2, $\mathcal{A}$ tries to learn as much as possible about L and $\mathcal{S}$. In step 3, verification is then simulated (for free) on all possible messages and signatures in order to find a forgery. However, this simulation only works sufficiently well if enough knowledge about L and $\mathcal{S}$ has been gathered in step 2.

Learning L and simulating with L. At any point in time, L should indeed only contain linear relations (also called "constraints") about $\mathfrak{g}$, i.e., $\mathbf{v}^{\mathsf{T}}\mathfrak{g} = \mathfrak{o}$ should be satisfied for every $\mathbf{v} \in L$. Such a relation is learned whenever $\mathcal{A}$ makes a call to its group-equality oracle that returns 1: assuming $\mathcal{A}$ kept track of the representations $\mathbf{t}_1$ and $\mathbf{t}_2$ of the two input group elements $\mathbf{t}_1^{\mathsf{T}}\mathfrak{g}$ and $\mathbf{t}_2^{\mathsf{T}}\mathfrak{g}$, it can update $L \leftarrow L + \langle \mathbf{v} \rangle$ to contain the new relation $\mathbf{v} := \mathbf{t}_1 - \mathbf{t}_2$.

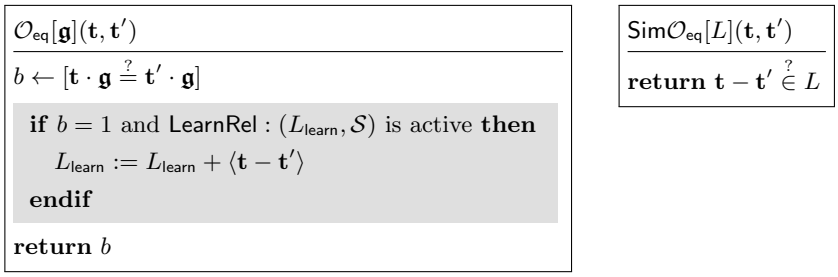

Fig. 3: Group equality oracles used within our adversary. **Left-hand side:** $\mathcal{O}_{\mathsf{eq}}[\mathfrak{g}]$ should be thought of as the *real* equality oracle; invoking it is costly. The shaded part is only used for learning whenever $\mathsf{LearnRel} : (L_{\mathsf{learn}}, \mathcal{S})$ is active (which we will always specify when invoking a procedure that utilizes $\mathcal{O}_{\mathsf{eq}}[\mathfrak{g}]$), and does not affect the outcome of the call to $\mathcal{O}_{\mathsf{eq}}[\mathfrak{g}]$. It ensures that all observed relation will be added to L_{learn}. **Right-hand side:** $\mathsf{Sim}\mathcal{O}_{\mathsf{eq}}[L]$ should be thought of as a *simulating* equality oracle, which is free.

To formalize this idea, we define the oracle $\mathcal{O}_{\mathsf{eq}}[\mathfrak{g}]$ in Figure 3, which takes two vectors $\mathbf{t}, \mathbf{t}' \in \mathbb{Z}_p^\infty$ as input and checks (using $\mathcal{A}$'s own group-equality oracle) whether $\mathbf{t}^{\mathsf{T}}\mathfrak{g}$ and $\mathbf{t}'^{\mathsf{T}}\mathfrak{g}$ are equal. Whenever the answer is positive, the shaded part of $\mathcal{O}_{\mathsf{eq}}[\mathfrak{g}]$ ensures that the corresponding relation is recorded. Note that we can plug in $\mathcal{O}_{\mathsf{eq}}[\mathfrak{g}]$ to any execution of $\widetilde{\mathsf{User}}_1$, $\widetilde{\mathsf{User}}_2$, or $\widetilde{\mathsf{Verify}}$ in order to make sure that all relations observed during the execution of these algorithms are stored. We do so during $\mathcal{A}$'s *learning phase*.

On the right-hand side in Figure 3, we also define an oracle $\mathsf{Sim}\mathcal{O}_{\mathsf{eq}}[L]$ that we plug into any execution of $\widetilde{\mathsf{User}}_1$, $\widetilde{\mathsf{User}}_2$, or $\widetilde{\mathsf{Verify}}$ that we want to *simulate* during the *forging phase* of $\mathcal{A}$. $\mathsf{Sim}\mathcal{O}_{\mathsf{eq}}[L]$

is entirely free, as it simply checks whether $\mathbf{t} - \mathbf{t}' \in L$, i.e., whether it "believes" that the real response would be 1.

Learning $\mathcal{S}$ and simulating with $\mathcal{S}$. Similarly to L, the adversary attempts to learn all important RO input-output pairs $\mathcal{S}$ before forging. Each member $(M, x, y) \in \mathcal{S}$ (with $M \in \mathbb{Z}_p^{n^{\mathsf{H}} \times \infty}$) indicates $\mathcal{H}(M \cdot \mathbf{g}, x) = y$. Such a triple is learned whenever $\mathcal{A}$ makes a call to its RO oracle $\mathcal{O}_{\mathsf{H}}$ on some input $(M \cdot \mathbf{g}, x)$ for some representation matrix M that it kept track of, and receives the output y: in this case, $\mathcal{A}$ can add (M, x, y) to $\mathcal{S}$.

$\mathcal{O}_{\mathsf{H}}^{\mathcal{H}}[\mathbf{g}](M, x)$

if $\mathsf{LearnRel} : (L_{\mathsf{learn}}, \mathcal{S})$ is active **then**
  for all $(M', x, y) \in \mathcal{S}$ and $i \in [n^{\mathsf{H}}]$ **do**
    if $M_i \cdot \mathbf{g} = M'_i \cdot \mathbf{g}$ **then**
      $L_{\mathsf{learn}} := L_{\mathsf{learn}} + \mathrm{rowsp}(M - M')$
    endif
  endfor
endif

$y \leftarrow \mathcal{H}(M \cdot \mathbf{g}, x)$

if $\mathsf{StoreRO} : \mathcal{S}_{\mathsf{learn}}$ is active **then**
  $\mathcal{S}_{\mathsf{learn}} := \mathcal{S}_{\mathsf{learn}} \cup \{(M, x, y)\}$
endif

return y

$\mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L, \mathcal{S}](M, x)$

if $\exists (M', x, y) \in \mathcal{S}$ with $\mathrm{rowsp}(M' - M) \subseteq L$ **then**
  return y
endif
return $\perp$

Fig. 4: These oracles provide the analogue of Figure 3 for the RO. **Left-hand side:** $\mathcal{O}_{\mathsf{H}}^{\mathcal{H}}[\mathbf{g}]$ behaves like the *real* RO; invoking it is costly. The shaded part again denotes the learning process. If we specify $\mathsf{StoreRO} : \mathcal{S}_{\mathsf{learn}}$ when invoking a procedure that utilizes $\mathcal{O}_{\mathsf{H}}^{\mathcal{H}}$, then all RO queries that are made will also be added to $\mathcal{S}_{\mathsf{learn}}$. Furthermore, as in Figure 3, whenever $\mathsf{LearnRel} : (L_{\mathsf{learn}}, \mathcal{S})$ is activated, we learn new linear relations by comparing to existing queries in $\mathcal{S}$. **Right-hand side:** The *simulation* counterpart to $\mathcal{O}_{\mathsf{H}}^{\mathcal{H}}[\mathbf{g}]$; it returns $\perp$ if no matching query was found.

To formalize this, we define the oracle $\mathcal{O}_{\mathsf{H}}^{\mathcal{H}}[\mathbf{g}]$ in Figure 4, which behaves like the *real* RO, but also ensures that all observed input-output pairs are recorded in $\mathcal{S}$ (in addition, using costly group equality operations, it checks whether a given RO input has been seen before by comparing it to all existing entries; if so, the corresponding linear relations on the RO input group elements are recorded as well). The "simulated" RO $\mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L, \mathcal{S}]$ is defined on the right-hand side of Figure 4. Note that it returns $\perp$ if no matching query was found in $\mathcal{S}$.

In Figure 5 we give two additional versions of $\mathcal{O}_{\mathsf{H}}$: The first is $\mathsf{Random}\mathcal{O}_{\mathsf{H}}$, which always returns a random output. Note that the combination $\mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L, \mathcal{S}] \prec \mathsf{Random}\mathcal{O}_{\mathsf{H}}$ essentially simulates a random oracle (consistent with $\mathcal{S}$) for free, so this is what $\mathcal{A}$ uses in the forging phase when trying to simulate the user/verification.

The second is $\mathsf{Randomize}\mathcal{O}_{\mathsf{H}}^{\mathcal{H}}[\mathbf{g}, \mathcal{S}]$, which is used in the learning phase. This oracle, on every input it receives, tosses a coin to decide whether to query the real RO or output a random value. This will be crucial when arguing that sufficient information about L and $\mathcal{S}$ is learned in the learning phase.

The learning phase. The full learning phase is defined in Figure 6. Each learning iteration i consists of sampling a random message m_i, computing the corresponding user-message $\widetilde{\mathsf{msgU}}_i$, querying for the signer-message $\widetilde{\mathsf{msgS}}_i$, and invoking GatherKnowledgeRand repeatedly. The latter repeatedly

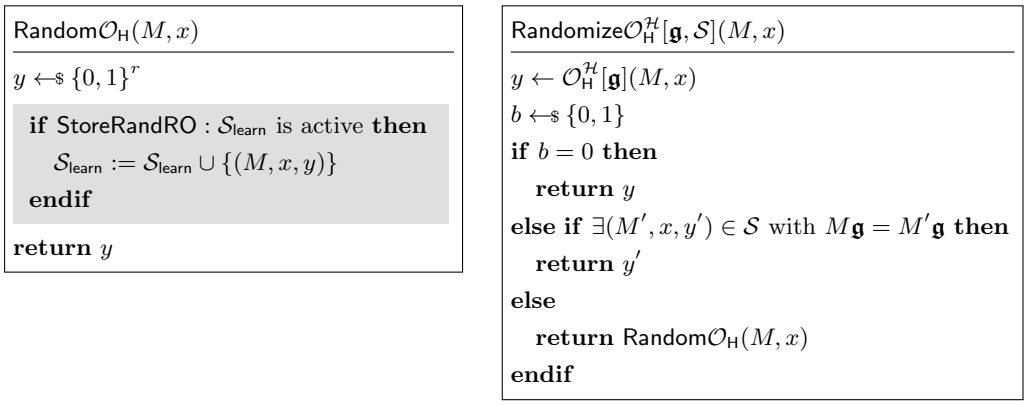

Fig. 5: Left-hand side: $\mathsf{Random}\mathcal{O}_{\mathsf{H}}$ always returns a random output. **Right-hand side:** $\mathsf{Randomize}\mathcal{O}^{\mathcal{H}}_{\mathsf{H}}[\mathfrak{g}, \mathcal{S}]$ is used inside GatherKnowledgeRand in Figure 6. It employs either the real RO or the random one with probability $\frac{1}{2}$ each. This will be crucial in the learning phase.

runs $\widetilde{\mathsf{User}_2}$ and $\widetilde{\mathsf{Verify}}$ with a re-randomized RO (as described in the overview, this is necessary to learn all information required when forging).

Note that we perform ℓ number of learning iterations, where ℓ is randomly chosen from $\{0, \ldots, \ell_{\mathsf{max}}\}$, with ℓ_{max} defined in Figure 6. The purpose of this is to simplify the proof: there are $\ell_{\mathsf{max}}/4$ "pieces of information" (see Definition 3.13) that we can learn that may be required for forging. In each iteration, we will (w.h.p.) either (1) learn a new piece of information, or (2) the existing knowledge is already sufficient for forging. By choosing ℓ at random, we can claim that with probability $\frac{3}{4}$, the message $m_{\ell+1}$ that we attempt to find a forgery for corresponds to case (2).

We formalize the desired outcome of the learning phase in Lemma 3.8.

The forging phase. $\mathcal{A}$ will then attempt to come up with a signer's message $\mathsf{msgS} = (\mathbf{u}^{\mathsf{msgS}}, s^{\mathsf{msgS}})$ that yields a valid signature for $m_{\ell+1}$. It does so by iterating through all possible tuples $(L_{\mathsf{forgery}}, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}})$[14] and simulating $\widetilde{\mathsf{User}_2}$ and $\widetilde{\mathsf{Verify}}$ with these values in order to check whether verification would succeed. The meaning of these four things is as follows:

- $L_{\mathsf{forgery}} \subseteq \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}+n^{\mathsf{msgS}}} \rangle$ should be thought of as a vector space of linear relations that the length-$(n^{\mathfrak{g}} + n^{\mathsf{msgS}})$ vector $\mathfrak{g}[1, \ldots, n^{\mathfrak{g}}] \| \mathbf{u}^{\mathsf{msgS}}$ will fulfill.
- s^{msgS} is an arbitrary bitstring that will be placed into msgS, verbatim.
- $\mathcal{S}_{\mathsf{fake}}$ is a set of "imaginary" RO input-outputs. When forging, User_2 is allowed to answer its queries using their "fake" output as determined by $\mathcal{S}_{\mathsf{fake}}$ (since a forgery σ does not need to be computed using an *honest* User_2), while Verify will have to work correctly without $\mathcal{S}_{\mathsf{fake}}$.
- We also utilize a set L_{fake} of "fake" linear relations for answering fake queries in $\mathcal{S}_{\mathsf{fake}}$. This allows the adversary to match some query that it makes during User_2 to some query in $\mathcal{S}_{\mathsf{fake}}$, even though the query's group elements use a different representation than that used by the element in $\mathcal{S}_{\mathsf{fake}}$.

If the simulated success probability for a tuple is sufficiently high, then Forge is run (which is not free anymore). This either results in a successful forgery, or $\mathcal{A}$ learns a new piece of information by doing so.

We prove that the forging phase is indeed successful (under the assumption that the learning phase was successful) in Lemma 3.9.

[14] L_{forgery} is similar to what is called L_{sim} in the overview (the difference being that L_{forgery} is a vector space on all of $\mathfrak{g}$, not just the group elements from vk). In addition, what was t in the overview is now called s^{msgS} in order to clarify its role among the large amount of other objects.

Learning Phase. Sample $\ell \leftarrow\!\!\$\ \{0,\ldots,\ell_{\mathsf{max}}\}$, where

$$\ell_{\mathsf{max}} := ((q_{\mathsf{KeyGen}} + \lceil \lambda/\alpha \rceil \cdot (q_{\mathsf{User}_1} + q_{\mathsf{Sign}} + q_{\mathsf{User}_2} + q_{\mathsf{Verify}})) \cdot (n^{\mathsf{H}} + 1) + 1 + n^{\mathsf{vk}}) \cdot 4$$
$$\beta = 1/2^{q+7} \quad \alpha = (\beta/(2q_{\mathsf{Verify}}(\lambda + q_{\mathsf{Sign}} + \log q_{\mathsf{Verify}})))^2$$

For each $i \in [\ell]$, do the following:

- $L, \mathcal{S}, m_i, \widetilde{\mathsf{st}}_i, \widetilde{\mathsf{msgU}}_i \leftarrow \textsc{InitSession}^{\mathcal{H}}(L, \mathcal{S}, \mathfrak{g}, \widetilde{\mathsf{vk}})$

  $\textsc{InitSession}^{\mathcal{H}}(L, \mathcal{S}, \mathfrak{g}, \widetilde{\mathsf{vk}})$

  sample $m \leftarrow\!\!\$\ \mathcal{M}$

  $\widetilde{\mathsf{st}}, \widetilde{\mathsf{msgU}} \leftarrow \widetilde{\mathsf{User}}_1^{\mathcal{O}_{\mathsf{eq}}[\mathfrak{g}], \mathcal{O}_{\mathsf{H}}^{\mathcal{H}}[\mathfrak{g}]}(\widetilde{\mathsf{vk}}, m)$

  with $\mathsf{LearnRel} : (L, \mathcal{S})$ *and* $\mathsf{StoreRO} : \mathcal{S}$ *activated*

  return $L, \mathcal{S}, m, \widetilde{\mathsf{st}}, \widetilde{\mathsf{msgU}}$

  This selects a random message and runs the first User on it, while keeping track of all linear relations in L and all RO triples in $\mathcal{S}$.

- $\mathfrak{g}, n^{\mathfrak{g}}, \widetilde{\mathsf{msgS}}_i \leftarrow \textsc{ObtainMsgS}^{\mathcal{O}_{\mathsf{Sign}}}(\mathfrak{g}, n^{\mathfrak{g}}, \widetilde{\mathsf{msgU}}_i)$

  $\textsc{ObtainMsgS}^{\mathcal{O}_{\mathsf{Sign}}}(\mathfrak{g}, n^{\mathfrak{g}}, \widetilde{\mathsf{msgU}} = (T^{\mathsf{msgU}}, s^{\mathsf{msgU}}))$

  $(\mathbf{u}^{\mathsf{msgS}}, s^{\mathsf{msgS}}) \leftarrow \mathcal{O}_{\mathsf{Sign}}(T^{\mathsf{msgU}} \cdot \mathfrak{g}, s^{\mathsf{msgU}})$

  $\mathfrak{g}[n^{\mathfrak{g}} + j] := \mathbf{u}_j^{\mathsf{msgS}}$ (for $j \in [n^{\mathsf{msgS}}]$) and $T^{\mathsf{msgS}} := (\mathbf{e}_{n^{\mathfrak{g}}+1} \mid \cdots \mid \mathbf{e}_{n^{\mathfrak{g}}+n^{\mathsf{msgS}}})^{\mathsf{T}}$

  $n^{\mathfrak{g}} := n^{\mathfrak{g}} + n^{\mathsf{msgS}}$

  return $\mathfrak{g}, n^{\mathfrak{g}}, \widetilde{\mathsf{msgS}} := (T^{\mathsf{msgS}}, s^{\mathsf{msgS}})$

  This queries the signing oracle. Newly received group elements are added to $\mathfrak{g}$, and T^{msgS} contains their descriptions pointing to those new elements.

- $(L_{\mathsf{new}}^{(j)}, \mathcal{S}_{\mathsf{new}}^{(j)}) \leftarrow \textsc{GatherKnowledgeRand}^{\mathcal{H}}(\mathfrak{g}, \mathcal{S}, \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}_i, \widetilde{\mathsf{msgS}}_i, m_i)$ for $j \in [k_{\mathsf{rand}}]$ with $k_{\mathsf{rand}} := \lambda \cdot 4 \cdot 2^q$, and after each iteration, update

$$L := L + L_{\mathsf{new}}^{(j)} \quad \text{and} \quad \mathcal{S} := \mathcal{S} \cup \mathcal{S}_{\mathsf{new}}^{(j)} .$$

  $\textsc{GatherKnowledgeRand}^{\mathcal{H}}(\mathfrak{g}, \mathcal{S}, \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}, \widetilde{\mathsf{msgS}}, m)$

  $L_{\mathsf{new}} := \{\mathbf{0}\}, \mathcal{S}_{\mathsf{new}} := \emptyset, \mathcal{S}_{\mathsf{rand}} := \emptyset$

  $\widetilde{\sigma} \leftarrow \widetilde{\mathsf{User}}_2^{\mathcal{O}_{\mathsf{eq}}[\mathfrak{g}], \mathsf{Randomize}\mathcal{O}_{\mathsf{H}}^{\mathcal{H}}[\mathfrak{g}, \mathcal{S}_{\mathsf{rand}}]}(\widetilde{\mathsf{st}}, \widetilde{\mathsf{msgS}})$

  $\widetilde{\mathsf{Verify}}^{\mathcal{O}_{\mathsf{eq}}[\mathfrak{g}], \mathsf{Randomize}\mathcal{O}_{\mathsf{H}}^{\mathcal{H}}[\mathfrak{g}, \mathcal{S}_{\mathsf{rand}}]}(\widetilde{\mathsf{vk}}, m, \widetilde{\sigma})$

  both $\widetilde{\mathsf{User}}_2$ *and* $\widetilde{\mathsf{Verify}}$ *have* $\mathsf{LearnRel} : (L_{\mathsf{new}}, \mathcal{S} \cup \mathcal{S}_{\mathsf{new}})$

  and $\mathsf{StoreRO} : \mathcal{S}_{\mathsf{new}}$ *and* $\mathsf{StoreRandRO} : \mathcal{S}_{\mathsf{rand}}$ *activated*

  return $(L_{\mathsf{new}}, \mathcal{S}_{\mathsf{new}})$

  This simulates each RO query with a fresh random output (instead of $\mathcal{H}$'s real output) with probability $\frac{1}{2}$, to ensure that we learn all necessary information.

Fig. 6: Our adversary's learning phase.

Forging Phase. Run $(L, \mathcal{S}, m_{\ell+1}, \widetilde{\mathsf{st}}_{\ell+1}, \widetilde{\mathsf{msgU}}_{\ell+1}) \leftarrow \textsc{InitSession}^{\mathcal{H}}(L, \mathcal{S}, \mathfrak{g}, \widetilde{\mathsf{vk}})$. Then, repeat the following $k_{\mathsf{attempts}} := 6 \cdot (n^{\mathfrak{g}} + q_{\mathsf{KeyGen}} + \ell \cdot q_{\mathsf{Sign}}) + 24\lambda$ many times:

- Find a tuple $(L_{\mathsf{forgery}}, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}})$ with $|\mathcal{S}_{\mathsf{fake}}| \leq q_{\mathsf{Sign}}$ and

$$L_{\mathsf{forgery}}, L_{\mathsf{fake}} \subseteq \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}+n^{\mathsf{msgS}}} \rangle \quad \text{and} \quad L_{\mathsf{forgery}} \cap \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}} \rangle \subseteq L$$

  s.t. $\Pr\left[\textsc{SimForge}(L + L_{\mathsf{forgery}}, \mathcal{S}, \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}_{\ell+1}, m_{\ell+1}, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}}) = 1\right] \geq \frac{1}{4}$.

$\textsc{SimForge}(L_{\mathsf{all}}, \mathcal{S}, \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}, m, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}})$

$\mathcal{S}_{\mathsf{rand}} := \emptyset$

$\widetilde{\mathsf{msgS}} := (T^{\mathsf{msgS}}, s^{\mathsf{msgS}})$ with $T^{\mathsf{msgS}} := (\mathbf{e}_{n^{\mathfrak{g}}+1} \mid \cdots \mid \mathbf{e}_{n^{\mathfrak{g}}+n^{\mathsf{msgS}}})^{\mathsf{T}}$

$\widetilde{\sigma} \leftarrow \widetilde{\mathsf{User}}_2^{\mathsf{Sim}\mathcal{O}_{\mathsf{eq}}[L_{\mathsf{all}}], \mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}}] \prec \mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L_{\mathsf{all}}, \mathcal{S} \cup \mathcal{S}_{\mathsf{rand}}] \prec \mathsf{Random}\mathcal{O}_{\mathsf{H}}}(\widetilde{\mathsf{st}}, \widetilde{\mathsf{msgS}})$

$b \leftarrow \widetilde{\mathsf{Verify}}^{\mathsf{Sim}\mathcal{O}_{\mathsf{eq}}[L_{\mathsf{all}}], \mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L_{\mathsf{all}}, \mathcal{S} \cup \mathcal{S}_{\mathsf{rand}}] \prec \mathsf{Random}\mathcal{O}_{\mathsf{H}}}(\widetilde{\mathsf{vk}}, m, \widetilde{\sigma})$

both with $\mathsf{StoreRandRO} : \mathcal{S}_{\mathsf{rand}}$ *activated*

return b

  This procedure (which does not require any real group operations or RO calls) tells us whether a potential tuple $(L_{\mathsf{forgery}}, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}})$ *is likely to succeed. The randomness inside the probability may come from* User_2 *itself, but more importantly also the "RO simulation"* $\mathsf{Random}\mathcal{O}_{\mathsf{H}}$.

- If no such tuple exists, **exit without forgery**. Otherwise, run

$$(L_{\mathsf{new}}, \mathcal{S}_{\mathsf{new}}, \sigma_{\ell+1}, b) \leftarrow \textsc{Forge}^{\mathcal{H}}(\mathfrak{g}, n^{\mathfrak{g}}, L, \mathcal{S}, \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}_{\ell+1}, m_{\ell+1}, L_{\mathsf{forgery}}, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}}) \ .$$

$\textsc{Forge}^{\mathcal{H}}(\mathfrak{g}, n^{\mathfrak{g}}, L, \mathcal{S}, \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}, m, L_{\mathsf{forgery}}, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}})$

$L_{\mathsf{new}} := \{\mathbf{0}\}, \mathcal{S}_{\mathsf{new}} := \emptyset$

compute A and B for which $L + L_{\mathsf{forgery}} = \mathrm{rowsp}(-B \mid A \mid 0 \ldots)$

sample $\mathbf{z} \leftarrow\!\!\$\ \mathbb{Z}_p^{n^{\mathsf{msgS}}}$

$\mathfrak{g}[n^{\mathfrak{g}} + 1 \ldots n^{\mathfrak{g}} + n^{\mathsf{msgS}}] := A^+ B \cdot \mathfrak{g}[1 \ldots n^{\mathfrak{g}}] + (A^+ A - I) \cdot \mathbf{z} \cdot \mathbf{1}$

$\widetilde{\mathsf{msgS}} := (T^{\mathsf{msgS}}, s^{\mathsf{msgS}})$ with $T^{\mathsf{msgS}} := (\mathbf{e}_{n^{\mathfrak{g}}+1} \mid \cdots \mid \mathbf{e}_{n^{\mathfrak{g}}+n^{\mathsf{msgS}}})^{\mathsf{T}}$

$(T^{\mathsf{sig}}, s^{\mathsf{sig}}) := \widetilde{\sigma} \leftarrow \widetilde{\mathsf{User}}_2^{\mathcal{O}_{\mathsf{eq}}[\mathfrak{g}], \mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}}] \prec \mathcal{O}_{\mathsf{H}}^{\mathcal{H}}[\mathfrak{g}]}(\widetilde{\mathsf{st}}, \widetilde{\mathsf{msgS}})$

$b \leftarrow \widetilde{\mathsf{Verify}}^{\mathcal{O}_{\mathsf{eq}}[\mathfrak{g}], \mathcal{O}_{\mathsf{H}}^{\mathcal{H}}[\mathfrak{g}]}(\widetilde{\mathsf{vk}}, m, \widetilde{\sigma})$

both with $\mathsf{LearnRel} : (L_{\mathsf{new}}, \mathcal{S} \cup \mathcal{S}_{\mathsf{new}})$ *and* $\mathsf{StoreRO} : \mathcal{S}_{\mathsf{new}}$ *activated*

return $(L_{\mathsf{new}}, \mathcal{S}_{\mathsf{new}}, \sigma, b)$ where $\sigma := (T^{\mathsf{sig}} \cdot \mathfrak{g}, s^{\mathsf{sig}})$

- **If** $b = 1$**, return** $((m_1, \sigma_1), \ldots, (m_{\ell+1}, \sigma_{\ell+1}))$, with $\sigma_1, \ldots, \sigma_\ell$ computed as

$$\sigma_i := (T_i^{\mathsf{sig}} \cdot \mathfrak{g}, s_i^{\mathsf{sig}}) \quad \text{with} \quad (T_i^{\mathsf{sig}}, s_i^{\mathsf{sig}}) \leftarrow \widetilde{\mathsf{User}}_2^{\mathcal{O}_{\mathsf{eq}}[\mathfrak{g}], \mathcal{O}_{\mathsf{H}}^{\mathcal{H}}[\mathfrak{g}]}(\widetilde{\mathsf{vk}}, m_i, \widetilde{\mathsf{msgS}}_i) \ .$$

  Otherwise, update $L := L + (L_{\mathsf{new}} \cap \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}} \rangle)$, $\mathcal{S} := \mathcal{S} \cup \mathcal{S}_{\mathsf{new}}$ and continue.

Fig. 7: Our adversary's forging phase.

3.3 Proof of Theorem 3.1

Towards proving that the adversary $\mathcal{A}$ as described in Section 3.2 breaks OMUF as required by Theorem 3.1, we first of all stress that the number of calls to $\mathcal{O}_{\mathsf{grp}}$, $\mathcal{O}_{\mathsf{eq}}$, $\mathcal{H}$, and $\mathcal{O}_{\mathsf{Sign}}$ made by $\mathcal{A}$ is indeed polynomial. (The number of signing queries is bounded by ℓ_{max} as defined in Figure 6.) The only part of $\mathcal{A}$ that is not poly-time is the first step of each iteration in the forging phase, since $\mathcal{A}$ needs to iterate through all possible tuples $(L_{\mathsf{forgery}}, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}})$. However, it does not make *any* oracle calls in there (see definition of SimForge).

We now give some definitions that will help modularizing our proof.

Definition 3.6. *We call a vector space $L \subseteq \mathbb{Z}_p^\infty$ of linear relations* valid *w.r.t. a vector $\mathbf{g}$ of group elements and $n^{\mathbf{g}} \in \mathbb{N}$ (s.t. $\mathbf{g}$ may be non-$\mathfrak{o}$ only in the first $n^{\mathbf{g}}$ components), if*

$$L \subseteq \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathbf{g}}} \rangle \quad \textit{and} \quad \forall \mathbf{v} \in L : \mathbf{v}^{\mathsf{T}} \cdot \mathbf{g} = \mathfrak{o} \; .$$

We call a set $\mathcal{S}$ of RO input-output pairs valid *w.r.t. RO $\mathcal{H}$, $\mathbf{g}$ and $n^{\mathbf{g}} \in \mathbb{N}$ (as above), if for all $(M, x, y) \in \mathcal{S}$:*

$$\mathrm{rowsp}(M) \subseteq \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathbf{g}}} \rangle \quad \textit{and} \quad \mathcal{H}(M \cdot \mathbf{g}, x) = y \; .$$

The following definition (when applied to message $m_{\ell+1}$) states what we hope from the *learning phase* to achieve (see Lemma 3.8):

Definition 3.7. *Given a current state of $\mathbf{g}$, $n^{\mathbf{g}}$, L, $\mathcal{S}$ and RO $\mathcal{H}$ (s.t. $\mathbf{g}$ is non-$\mathfrak{o}$ only in the first $n^{\mathbf{g}}$ components, and L and $\mathcal{S}$ are valid), as well as $\widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}, m$, we say that* $\mathsf{Forgeability}(\mathbf{g}, n^{\mathbf{g}}, L, \mathcal{S}, \mathcal{H}, \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}, m)$ *holds, if the following is true:*

There exists a tuple $(L_{\mathsf{forgery}}, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}})$ with $|\mathcal{S}_{\mathsf{fake}}| \leq q_{\mathsf{Sign}}$ and

$$L_{\mathsf{forgery}}, L_{\mathsf{fake}} \subseteq \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathbf{g}}+n^{\mathsf{msgS}}} \rangle \quad \textit{and} \quad L_{\mathsf{forgery}} \cap \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathbf{g}}} \rangle \subseteq L$$

s.t. for every valid $L' \supseteq L$ and for every valid $\mathcal{S}' \supseteq \mathcal{S}$:

$$\Pr\left[\textsc{SimForge}(L' + L_{\mathsf{forgery}}, \mathcal{S}', \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}, m, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}}) = 1\right] \geq \frac{1}{2} \; .$$

We stress the slight mismatch between the requirements that the definition above places on $(L_{\mathsf{forgery}}, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}})$, and the requirements that our adversary's *forging phase* places on the tuple $(L_{\mathsf{forgery}}, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}})$ it attempts to find: Definition 3.7 requires the probability over an execution of SimForge to be sufficiently large for *all valid* $L' \supseteq L$ and *all valid* $\mathcal{S}' \supseteq \mathcal{S}$, while our adversary only needs it to be sufficiently large for *the current state* of L and $\mathcal{S}$. The reason for requiring the stronger Definition 3.7 is that during the forging phase, there may be failed attempts that need to be utilized for advancing the knowledge of L and $\mathcal{S}$. Hence, the latter may change throughout the forging phase.

The following two lemmas show that (1) the learning phase yields the desired outcome with sufficient probability, and (2) the desired outcome (i.e., Forgeability) indeed results in a successful forging phase. Combining Lemmas 3.8 and 3.9, we immediately get Theorem 3.1 (because the probability of the game $\mathbf{omuf}_{\mathsf{BS},\mathcal{A}}$ returning 1 becomes $\geq \frac{1}{2} - \mathrm{negl}(\lambda)$).

Lemma 3.8. *Given a blind signature scheme* BS *with the same restrictions as in Theorem 3.1, the following holds for sufficiently large λ:*

$$\Pr[\mathbf{omuf}_{\mathsf{BS},\mathcal{A}} : \mathsf{Forgeability}(\mathbf{g}, n^{\mathbf{g}}, L, \mathcal{S}, \mathcal{H}, \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}_{\ell+1}, m_{\ell+1})] \geq \frac{1}{2} \; ,$$

where $\mathbf{g}$, $n^{\mathbf{g}}$, L, and $\mathcal{S}$ refer to the state of the variables of the same name used within adversary $\mathcal{A}$, at the point in time immediately after running InitSession *at the beginning of the forging phase.*

Lemma 3.9. *Using the same notation as in Lemma 3.8, the following holds:*

$$\begin{aligned}&\Pr[\mathbf{omuf}_{\mathsf{BS},\mathcal{A}} \textit{ outputs } 1]\\ \geq\ &\Pr[\mathbf{omuf}_{\mathsf{BS},\mathcal{A}} : \mathsf{Forgeability}(\mathfrak{g}, n^{\mathfrak{g}}, L, \mathcal{S}, \mathcal{H}, \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}_{\ell+1}, m_{\ell+1})] - \mathrm{negl}(\lambda)\ .\end{aligned}$$

In Section 3.4, we give the outline for proving Lemma 3.8, following that we prove a range of lemmas required for Lemma 3.8, and finally in Section 3.9 we prove Lemma 3.9.

3.4 Outline for proving Forgeability (Lemma 3.8)

Note that in Lemma 3.8, we are not concerned at all with the *forging phase* of the adversary, but only with the outcome of the *learning phase*. To this end, we consider the experiment Exp defined in Figure 8.

1. $\mathcal{H}$ is a fresh RO. Run $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.
2. For $i \in [\lceil \lambda/\alpha \rceil]$ times, sample $m \leftarrow \mathcal{M}$, and run the entire protocol, i.e., $(\mathsf{st}, \mathsf{msgU}) \leftarrow \mathsf{User}_1(\mathsf{vk}, m)$, $\mathsf{msgS} \leftarrow \mathsf{Sign}(\mathsf{sk}, \mathsf{msgU})$, $\sigma \leftarrow \mathsf{User}_2(\mathsf{st}, \mathsf{msgS})$, and $\mathsf{Verify}(\mathsf{vk}, m, \sigma)$.
3. Run $\mathcal{A}(\mathsf{vk})$'s initialization step to create $\mathfrak{g}_0$, $n_0^{\mathfrak{g}}$, and $\widetilde{\mathsf{vk}}$. Define $L_0^* := \{\mathbf{0}\}$ and $\mathcal{S}_0^* := \emptyset$.
4. Sample $\ell \leftarrow \{0, \ldots, \ell_{\mathsf{max}}\}$, but run the following for each $i \in [\ell_{\mathsf{max}} + 1]$:
   - $(L_i, \mathcal{S}_i, m_i, \widetilde{\mathsf{st}}_i, \widetilde{\mathsf{msgU}}_i) \leftarrow \textsc{InitSession}^{\mathcal{H}}(L_{i-1}^*, \mathcal{S}_{i-1}^*, \mathfrak{g}_{i-1}, \widetilde{\mathsf{vk}})$
   - $\mathfrak{g}_i, n_i^{\mathfrak{g}}, \widetilde{\mathsf{msgS}}_i \leftarrow \textsc{ObtainMsgS}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(\mathfrak{g}_{i-1}, n_{i-1}^{\mathfrak{g}}, \widetilde{\mathsf{msgU}}_i)$
   - Initialize $L_{i,0} := L_i$ and $\mathcal{S}_{i,0} := \mathcal{S}_i$, and for each $j \in [k_{\mathsf{rand}}]$, run
   $$(L_{\mathsf{new}}^{(j)}, \mathcal{S}_{\mathsf{new}}^{(j)}) \leftarrow \textsc{GatherKnowledgeRand}^{\mathcal{H}}(\mathfrak{g}, \mathcal{S}_{i,j-1}, \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}_i, \widetilde{\mathsf{msgS}}_i, m_i)$$
   and update $L_{i,j} := L_{i,j-1} + L_{\mathsf{new}}^{(j)}$ and $\mathcal{S}_{i,j} := \mathcal{S}_{i,j-1} \cup \mathcal{S}_{\mathsf{new}}^{(j)}$.
   Then choose $L_i^* := L_{i,k_{\mathsf{rand}}}$ and $\mathcal{S}_i^* := \mathcal{S}_{i,k_{\mathsf{rand}}}$.

Fig. 8: The experiment Exp. We note the following differences compared to $\mathbf{omuf}_{\mathsf{BS},\mathcal{A}}$: (1) we omit the forging phase, (2) the learning phase is run for exactly $\ell_{\mathsf{max}}+1$ iterations (i.e., even for the message $m_{\ell+1}$ that $\mathcal{A}$ would attempt to forge a signature for), (3) we keep track of the state of L and $\mathcal{S}$ in each iteration (denoted by L_i and $\mathcal{S}_i$), and (4) step 2 (which runs a large number of signing sessions that does not have any effect on the remainder) is entirely new and merely used to help us in the proof.

Comparing Exp with $\mathbf{omuf}_{\mathsf{BS},\mathcal{A}}$, note that $\mathcal{S}_i$ and L_i can be thought of as the state that $\mathcal{S}$ and L would have in $\mathbf{omuf}_{\mathsf{BS},\mathcal{A}}$ *right before the* k_{rand} *rerandomizations are performed* in the i-th iteration, and $\mathcal{S}_i^*$ and L_i^* is their state *afterwards*. Thus, $L_{\ell+1}$ and $\mathcal{S}_{\ell+1}$ correspond to the state of variables L and $\mathcal{S}$ in the first iteration of the forging phase. We always have $L_{i+1} \subseteq \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n_i^{\mathfrak{g}}} \rangle$ and $L_{i+1}^* \subseteq \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n_{i+1}^{\mathfrak{g}}} \rangle$.

Lemma 3.8 is immediately implied by

$$\Pr[\mathsf{Exp} : \mathsf{Forgeability}(\mathfrak{g}_\ell, n_\ell^{\mathfrak{g}}, L_{\ell+1}, \mathcal{S}_{\ell+1}, \mathcal{H}, \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}_{\ell+1}, m_{\ell+1})] \geq \frac{1}{2}\ , \tag{11}$$

since we did not make any changes to $\mathbf{omuf}_{\mathsf{BS},\mathcal{A}}$ that can have an effect on $\mathsf{Forgeability}$, i.e., the probability above is identical to that in Lemma 3.8.

Definitions regarding Exp. In the experiment Exp, we denote by $\mathcal{R}$ the set of all RO queries $(\mathfrak{h}, x)$ that have been made at some point during any of the 5 algorithms in steps 1 and 2. (Intuitively, this set $\mathcal{R}$ will contain w.h.p. all irrelevant queries, which then allows us to use our result concerning irrelevant queries, see Lemma 3.4). Similarly, we define the set $\mathcal{Q}_i^{\mathsf{User}_1}$ (resp. $\mathcal{Q}_i^{\mathsf{Sign}}$) for each $i \in \{0, \dots, \ell_{\mathsf{max}}\}$ as the set of RO queries $(\mathfrak{h}, x)$ that have been made at some point during the i-th execution of InitSession (resp. ObtainMsgS).

Hybrids. We prove (11) by relating SimForge (as defined in the forging phase, see Figure 7) with the execution of $\widetilde{\mathsf{User}}_2$ and $\widetilde{\mathsf{Verify}}$ on the *real* signer-message $\mathsf{msgS}_{\ell+1}$ that is present in Exp. To this end, we define the two hybrid procedures in Figure 9 and corresponding shorthands

$$\mathrm{Hyb}_0^i := \mathrm{Hyb}_0(\mathfrak{g}_i, \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}_i, m_i, \widetilde{\mathsf{msgS}}_i, \mathcal{R}, \mathcal{Q}_i^{\mathsf{Sign}}, \mathcal{Q}_i^{\mathsf{User}_1}, \mathcal{H}) \quad \text{and}$$
$$\mathrm{Hyb}_1^i(L'_{\mathsf{all}}, \mathcal{S}) := \mathrm{Hyb}_1(\mathfrak{g}_i, \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}_i, m_i, \widetilde{\mathsf{msgS}}_i, \mathcal{R}, \mathcal{Q}_i^{\mathsf{Sign}}, \mathcal{Q}_i^{\mathsf{User}_1}, \mathcal{H}, L'_{\mathsf{all}}, \mathcal{S}')$$

for each $i \in \{0, \dots, \ell_{\mathsf{max}} + 1\}$.

Definition 3.10. *We define an event for both hybrids from Figure 9:*

- BadVerifyQuery*:* $\widetilde{\mathsf{Verify}}$ *at any point made a query* (M, x) *with* $(M\mathfrak{g}, x) \in \mathcal{Q}^{\mathsf{Sign}} \setminus \mathcal{R}$.

In Hyb_1*, we further define the following event:*

- InconsistentRO*: Two queries* (M, x) *and* (M', x) *with* $M\mathfrak{g} = M'\mathfrak{g}$ *have been made (either by* $\widetilde{\mathsf{User}}_2$ *or by* $\widetilde{\mathsf{Verify}}$*, or each algorithm made one of the two queries) for which two* different *outputs* $y \neq y'$ *were returned.*

Note that InconsistentRO only applies Hyb_1, because it can never be true inside Hyb_0 (since $\mathcal{O}_{\mathsf{H}}^{\mathcal{H}'}[\mathfrak{g}](M, x)$ returns $\mathcal{H}'(M\mathfrak{g}, x)$, with $\mathcal{H}'$ being a *function*).

$\mathrm{Hyb}_0(\mathfrak{g}, \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}, m, \widetilde{\mathsf{msgS}}, \mathcal{R}, \mathcal{Q}^{\mathsf{Sign}}, \mathcal{Q}^{\mathsf{User}_1}, \mathcal{H})$

$\mathcal{H}'$ is a fresh RO, except that $\forall (\mathfrak{h}, x) \in \mathcal{R} \cup \mathcal{Q}^{\mathsf{User}_1} \cup \mathcal{Q}^{\mathsf{Sign}} : \mathcal{H}'(\mathfrak{h}, x) = \mathcal{H}(\mathfrak{h}, x)$

$\widetilde{\sigma} \leftarrow \widetilde{\mathsf{User}}_2^{\mathcal{O}_{\mathsf{eq}}[\mathfrak{g}], \mathcal{O}_{\mathsf{H}}^{\mathcal{H}'}[\mathfrak{g}]}(\widetilde{\mathsf{st}}, \widetilde{\mathsf{msgS}})$

$b \leftarrow \widetilde{\mathsf{Verify}}^{\mathcal{O}_{\mathsf{eq}}[\mathfrak{g}], \mathcal{O}_{\mathsf{H}}^{\mathcal{H}'}[\mathfrak{g}]}(\widetilde{\mathsf{vk}}, m, \widetilde{\sigma})$

$\mathrm{Hyb}_1(\mathfrak{g}, \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}, m, \widetilde{\mathsf{msgS}}, \mathcal{R}, \mathcal{Q}^{\mathsf{Sign}}, \mathcal{Q}^{\mathsf{User}_1}, \mathcal{H}, L'_{\mathsf{all}}, \mathcal{S}')$

$\mathcal{H}'$ is a fresh RO, except that $\forall (\mathfrak{h}, x) \in \mathcal{R} \cup \mathcal{Q}^{\mathsf{User}_1} \cup \mathcal{Q}^{\mathsf{Sign}} : \mathcal{H}'(\mathfrak{h}, x) = \mathcal{H}(\mathfrak{h}, x)$

$\widetilde{\sigma} \leftarrow \widetilde{\mathsf{User}}_2^{\mathcal{O}_{\mathsf{eq}}[\mathfrak{g}], \mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L'_{\mathsf{all}}, \mathcal{S}'] \prec \mathcal{O}_{\mathsf{H}}^{\mathcal{H}'}[\mathfrak{g}]}(\widetilde{\mathsf{st}}, \widetilde{\mathsf{msgS}})$

$b \leftarrow \widetilde{\mathsf{Verify}}^{\mathcal{O}_{\mathsf{eq}}[\mathfrak{g}], \mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L'_{\mathsf{all}}, \mathcal{S}'] \prec \mathcal{O}_{\mathsf{H}}^{\mathcal{H}'}[\mathfrak{g}]}(\widetilde{\mathsf{vk}}, m, \widetilde{\sigma})$

Fig. 9: Hybrids that $\mathrm{SimForge}(L' + L_{\mathsf{forgery}}, \mathcal{S}', \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}_{\ell+1}, m_{\ell+1}, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}})$ will be close to. The difference between the two hybrids is highlighted using a shaded background. Hyb_0 will have high success probability by correctness and blindness, and hence SimForge will also have high success probability.

We start by showing that $\mathrm{Hyb}_0^{\ell+1}$ has indeed large success probability, and that furthermore it is unlikely that Verify would make an RO query that coincides with one of the signers queries (unless

already in $\mathcal{R}$ or previously made by User_1). The former follows by the scheme's correctness, and the latter from our result about irrelevant queries (Lemma 3.4). The following, proven in Section 3.5, summarizes this:

Lemma 3.11. *In the experiment* Exp*, we say that the event* Bad_0 *holds whenever*

$$\Pr\left[\mathrm{HYB}_0^{\ell+1} : b = 0 \vee \mathsf{BadVerifyQuery}\right] > \frac{1}{2^{q+4}} .$$

For sufficiently large λ*, we have* $\Pr[\mathsf{Exp} : \mathsf{Bad}_0] \leq \frac{1}{4}$*.*[15]

In the next step, we show that HYB_0 and HYB_1 are close. Intuitively, since HYB_1 utilizes the simulating RO $\mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L'_{\mathsf{all}}, \mathcal{S}']$ before invoking $\mathcal{O}_{\mathsf{H}}^{\mathcal{H}'}[\mathfrak{g}]$ (given some "knowledge" $\mathcal{S}'$ about the real RO $\mathcal{H}$), this can be interpreted as the fact that an increasing amount of knowledge about $\mathcal{H}$ does not hurt the success probability by too much.[16] In other words, while HYB_0 assigns fresh outputs to all RO queries except those in $\mathcal{R} \cup \mathcal{Q}^{\mathsf{User}_1} \cup \mathcal{Q}^{\mathsf{Sign}}$, HYB_1 treats *more* RO queries as fixed and assigns only fresh outputs to those that are not already covered by what $\mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L'_{\mathsf{all}}, \mathcal{S}']$ knows about $\mathcal{H}$. This is captured by the following lemma (proven in Section 3.6).

Lemma 3.12. *In the experiment* Exp*, we say that the event* Bad_1 *holds whenever there exists a valid vector space* $L'_{\mathsf{all}} \subseteq \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}_{\ell+1}} \rangle$ *with* $L'_{\mathsf{all}} \supseteq L_{\ell+1}$ *and a valid set* $\mathcal{S}' \supseteq \mathcal{S}_{\ell+1}$*, s.t.*

$$\begin{aligned} &\Pr\left[\mathrm{HYB}_1^{\ell+1}(L'_{\mathsf{all}}, \mathcal{S}') : (b = 0 \vee \mathsf{BadVerifyQuery}) \wedge \overline{\mathsf{InconsistentRO}}\right] \\ &> 2^{q+2} \cdot \Pr\left[\mathrm{HYB}_0^{\ell+1} : b = 0 \vee \mathsf{BadVerifyQuery}\right] . \end{aligned}$$

We have $\Pr[\mathsf{Exp} : \mathsf{Bad}_1] \leq \frac{1}{4}$*.*

Measuring knowledge. The connection between HYB_1 and SimForge is the most involved part of the proof of Lemma 3.8, because we will need to prove that the adversary indeed learns sufficient knowledge. To this end, we first need to precisely define what we mean by "knowledge", and we need a way to measure it: For each $i \in [\ell_{\mathsf{max}} + 1]$, we define the vector space

$$V_i := \langle \mathbf{e}_1, \ldots, \mathbf{e}_{1+n^{\mathsf{vk}}} \rangle + \sum_{\substack{(M,x,y) \in \mathcal{S}_i \\ (M \cdot \mathfrak{g}, x) \in \mathcal{R}}} \mathrm{rowsp}(M) .$$

Note that this holds all descriptions of "necessary" group elements that were seen up to iteration i. "Necessary" here means we may require them for forging; this includes the generator $\mathtt{1}$, the verification key's elements, and those group elements that are part of any input to an RO query contained in the set $\mathcal{R}$.

For the experiment Exp, we now define the event $\mathsf{MissingKnowledge}_i$. Intuitively, this event states that for message m_i and corresponding state $\widetilde{\mathsf{st}}_i$ and $\widetilde{\mathsf{msgS}}_i$, when running $\mathsf{User}_2/\mathsf{Verify}$, it is likely (i.e., has probability $\geq \frac{1}{4}$) that we learn new relations about the necessary group elements (i.e., those described by V_i) *or* that we encounter at least one new query in $\mathcal{R}$ that we have never seen before (which may enlargen the vector space V_i). For our adversary, $\mathsf{MissingKnowledge}_i$ would mean that we are unable to find a forgery for m_i with sufficient probability, and therefore we want $\mathsf{MissingKnowledge}_{\ell+1}$ to be false. On the other hand, whenever $\mathsf{MissingKnowledge}_i$ is true, then we will be able to show that we likely learn new knowledge in the i-th iteration.

[15] We point out that Bad_0 is an event in the experiment Exp, but its definition contains a probability itself. That probability is taken over randomness inside $\mathrm{HYB}_0^{\ell+1}$ (such as the fresh RO $\mathcal{H}'$ generated by $\mathrm{HYB}_0^{\ell+1}$) that is entirely separate from the randomness that is used in Exp.

[16] It may sound obvious that more knowledge can never be worse than less knowledge. However, this is not quite true in our case, where every piece of knowledge may affect our simulated forging probability.

Definition 3.13. *In* HYB_1*, we can initialize* $L_{\mathsf{new}} := \{\mathbf{0}\}, \mathcal{S}_{\mathsf{new}} := \emptyset$*, and run both* $\widetilde{\mathsf{User}}_2$ *and* $\widetilde{\mathsf{Verify}}$ *with* $\mathsf{LearnRel} : (L_{\mathsf{new}}, \mathcal{S} \cup \mathcal{S}_{\mathsf{new}})$ *and* $\mathsf{StoreRO} : \mathcal{S}_{\mathsf{new}}$ *activated. This does not change anything about the distribition of existing variables.*

Given this, we define the following events for HYB_1^i*:*

- $\mathsf{LearnedRelation}$*:* $L_{\mathsf{new}} \cap V_i \not\subseteq L_i$*.*
- $\mathsf{LearnedQuery}$*: There exists a triple* $(\mathfrak{h}, x) \in \mathcal{R}$ *s.t. there is no* $(M, x, y) \in \mathcal{S}_i$ *with* $M\mathfrak{g} = \mathfrak{h}$*, but there is some* $(M, x, y') \in \mathcal{S}_{\mathsf{new}}$ *with* $M\mathfrak{g} = \mathfrak{h}$*.*

For the experiment Exp*, we say that the event* $\mathsf{MissingKnowledge}_i$ *(for* $i \in [\ell_{\mathsf{max}} + 1]$*) holds if there exists a valid space of relations* $L'_{\mathsf{all}} \subseteq \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}_{i+1}} \rangle$ *with* $L_i \subseteq L'_{\mathsf{all}}$*, as well as a valid set of RO triples* $\mathcal{S}' \supseteq \mathcal{S}_i$*, s.t.*

$$\Pr[\text{HYB}_1^i(L'_{\mathsf{all}}, \mathcal{S}') : \mathsf{LearnedRelation} \vee \mathsf{LearnedQuery}] \geq \frac{1}{4} \,. \tag{12}$$

Now, connecting $\text{HYB}_1^{\ell+1}$ to SIMFORGE consists of two parts: First, event $\mathsf{MissingKnowledge}_{\ell+1}$ is unlikely to hold, captured by the following lemma proven in Section 3.7. It mainly uses the observation that $\mathcal{A}$ repeatedly invoking $\text{GATHERKNOWLEDGERAND}$ can be seen as running HYB_1 for *every* choice of $L', L_{\mathsf{fake}}, \mathcal{S}'$, and $\mathcal{S}_{\mathsf{fake}}$ (even though there are exponentially many of them).

Lemma 3.14. *The following holds for sufficiently large* λ*:*

$$\Pr[\mathsf{Exp} : \mathsf{MissingKnowledge}_{\ell+1}] < \frac{1}{4} \,.$$

Finally, provided that $\overline{\mathsf{MissingKnowledge}_{\ell+1}}$ holds, the existing knowledge $L, \mathcal{S}$ it is indeed sufficient to *simulate* all relations and necessary RO calls. This is captured by the following lemma (proven in Section 3.8).

Lemma 3.15. *Consider an execution of the experiment* Exp *for which the event* $\overline{\mathsf{MissingKnowledge}_{\ell+1}}$ *holds. Then, there must exist a tuple* $(L_{\mathsf{forgery}}, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}})$ *with* $|\mathcal{S}_{\mathsf{fake}}| \leq q_{\mathsf{Sign}}$ *and*

$$L_{\mathsf{forgery}}, L_{\mathsf{fake}} \subseteq \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}_{\ell+1}} \rangle \quad \text{and} \quad L_{\mathsf{forgery}} \cap \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}_{\ell}} \rangle \subseteq L_{\ell+1}$$

s.t. for every valid $L' \subseteq \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}_{\ell}} \rangle$ *with* $L' \supseteq L_{\ell+1}$ *and for every valid* $\mathcal{S}' \supseteq \mathcal{S}_{\ell+1}$*, we have*

$$\begin{aligned} &\Pr[\text{SIMFORGE}(L' + L_{\mathsf{forgery}}, \mathcal{S}', \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}_{\ell+1}, m_{\ell+1}, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}}) \text{ outputs } 1] \\ \geq\ &\Pr[\text{HYB}_1^{\ell+1}(L' + L_{\mathsf{forgery}}, \mathcal{S}') : (b = 1 \wedge \overline{\mathsf{BadVerifyQuery}}) \vee \mathsf{InconsistentRO}] - \frac{1}{4} \,. \end{aligned}$$

Putting it together. Combining Lemmas 3.11, 3.12, 3.14 and 3.15, we get (for sufficiently large λ):

$$\begin{aligned} &\Pr[\mathsf{Exp} : \mathsf{Forgeability}(\mathfrak{g}_\ell, n^{\mathfrak{g}}_{\ell}, L_{\ell+1}, \mathcal{S}_{\ell+1}, \mathcal{H}, \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}_{\ell+1}, m_{\ell+1})] \\ \geq &\Pr[\overline{\mathsf{Bad}_0} \wedge \overline{\mathsf{Bad}_1} \wedge \overline{\mathsf{MissingKnowledge}_{\ell+1}}] \geq \frac{1}{4} \,, \end{aligned}$$

because $\overline{\mathsf{Bad}_0} \wedge \overline{\mathsf{Bad}_1} \wedge \overline{\mathsf{MissingKnowledge}_{\ell+1}}$ implies existence of $(L_{\mathsf{forgery}}, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}})$, s.t. for all valid $L' \subseteq \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}_{\ell+1}} \rangle$ with $L' \supseteq L_{\ell+1}$ and for every valid $\mathcal{S}' \supseteq \mathcal{S}_{\ell+1}$:

$$\begin{aligned} &\Pr[\text{SIMFORGE}(L' + L_{\mathsf{forgery}}, \mathcal{S}', \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}_{\ell+1}, m_{\ell+1}, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}}) \text{ outputs } 1] \\ \geq\ &(1 - 2^{q+2} \cdot \Pr\left[\text{HYB}_0^{\ell+1} : b = 0 \vee \mathsf{BadVerifyQuery}\right]) - \frac{1}{4} \geq \frac{1}{2} \end{aligned}$$

3.5 Proof of Lemma 3.11 (Success probability of $\mathbf{Hyb}_0^{\ell+1}$)

We first recall the lemma that we want to prove:

Lemma 3.11. *In the experiment* Exp*, we say that the event* Bad_0 *holds whenever*

$$\Pr\left[\mathrm{HYB}_0^{\ell+1} : b = 0 \vee \mathsf{BadVerifyQuery}\right] > \frac{1}{2^{q+4}} .$$

For sufficiently large λ*, we have* $\Pr[\mathsf{Exp} : \mathsf{Bad}_0] \leq \frac{1}{4}$.[17]

Proof. Suppose we are running both Exp and then $\mathrm{HYB}_0^{\ell+1}$ in sequence. By correctness of the blind signature scheme, the signature computed within $\mathrm{HYB}_0^{\ell+1}$ for the message $m_{\ell+1}$ that was generated by Exp must be valid with overwhelming probability, i.e.,

$$\Pr\left[\begin{matrix}\mathsf{Exp}\\ \mathrm{HYB}_0^{\ell+1}\end{matrix} : b = 1\right] \geq 1 - \mathrm{negl}(\lambda) .$$

Recall that b refers to the output of $\widetilde{\mathsf{Verify}}$ invoked inside $\mathrm{HYB}_0^{\ell+1}$. While Exp and $\mathrm{HYB}_0^{\ell+1}$ run the blind signature algorithms using different RO's, correctness still applies: First, Exp runs KeyGen, User_1, and Sign using $\mathcal{H}$. Then, $\mathrm{HYB}_0^{\ell+1}$ samples a fresh RO $\mathcal{H}'$, except it enforces that $\mathcal{H}'$ matches $\mathcal{H}$ on all inputs in $\mathcal{R} \cup \mathcal{Q}^{\mathsf{User}_1} \cup \mathcal{Q}^{\mathsf{Sign}}$. Hence, the entire execution is equivalent to using a *single* lazy RO, and therefore it is also equivalent to the experiment inside the correctness definition (Definition 2.5).

We let $\mathcal{Q}^{\mathsf{Verify}}$ be the set of RO queries $(\mathfrak{h}, x)$ made within the execution of $\widetilde{\mathsf{Verify}}$ inside $\mathrm{HYB}_0^{\ell+1}$. Then, by Lemma 3.4, with probability at most β, verification makes an irrelevant query that the signer has made as well, i.e.,

$$\Pr\left[\begin{matrix}\mathsf{Exp}\\ \mathrm{HYB}_0^{\ell+1}\end{matrix} : b = 1 \wedge (\mathcal{Q}^{\mathsf{Verify}} \cap \mathcal{Q}^{\mathsf{Sign}}_{\ell+1}) \setminus \mathcal{IRR}^{\mathsf{vk},\mathsf{sk}}_{\alpha,\mathcal{H}} = \emptyset\right] \geq 1 - \mathrm{negl}(\lambda) - \beta . \tag{13}$$

Otherwise there would be a polynomial-time blindness adversary with non-negligible advantage.

Also note that in the experiment Exp, we have

$$\Pr[\mathsf{Exp} : \mathcal{R} \supseteq \mathcal{IRR}^{\mathsf{vk},\mathsf{sk}}_{\alpha,\mathcal{H}}] \geq 1 - \mathrm{negl}(\lambda) . \tag{14}$$

This follows from the following observation: Recall that the set $\mathcal{R}$ is computed in Exp through $\lceil \lambda/\alpha \rceil$ independent signing sessions. Thus, for each individual irrelevant query $(\mathfrak{h}, x) \in \mathcal{IRR}^{\mathsf{vk},\mathsf{sk}}_{\alpha,\mathcal{H}}$, by definition (see Definition 3.2), we have

$$\Pr[\mathsf{Exp} : (\mathfrak{h}, x) \in \mathcal{R}] \geq 1 - (1-\alpha)^{\lambda/\alpha} \geq 1 - e^{-\lambda} .$$

Thus, (14) holds by union-bound (which also uses the fact that the size of $\mathcal{IRR}^{\mathsf{vk},\mathsf{sk}}_{\alpha,\mathcal{H}}$ is polyomial, see Lemma 3.3).

Combining Equations (13) and (14) with $\beta = \frac{1}{2^{q+7}}$, we get

$$\Pr\left[\begin{matrix}\mathsf{Exp}\\ \mathrm{HYB}_0^{\ell+1}\end{matrix} : b = 1 \wedge (\mathcal{Q}^{\mathsf{Verify}} \cap \mathcal{Q}^{\mathsf{Sign}}_{\ell+1}) \setminus \mathcal{R} = \emptyset\right] \geq 1 - \mathrm{negl}(\lambda) - \beta \geq 1 - \frac{1}{2^{q+6}} ,$$

for sufficiently large λ. Note that, by definition, $(\mathcal{Q}^{\mathsf{Verify}} \cap \mathcal{Q}^{\mathsf{Sign}}_{\ell+1}) \setminus \mathcal{R} = \emptyset$ holds iff $\overline{\mathsf{BadVerifyQuery}}$. Thus, as desired, we get $\Pr[\mathsf{Exp} : \mathsf{Bad}_0] \leq \frac{1}{4}$ (since otherwise, taken over both Exp and $\mathrm{HYB}_0^{\ell+1}$, the probability of $b = 0 \vee \mathsf{BadVerifyQuery}$ would be at greater than $\frac{1}{2^{q+6}}$, thus contradicting the inequality above.) □

[17] We point out that Bad_0 is an event in the experiment Exp, but its definition contains a probability itself. That probability is taken over randomness inside $\mathrm{HYB}_0^{\ell+1}$ (such as the fresh RO $\mathcal{H}'$ generated by $\mathrm{HYB}_0^{\ell+1}$) that is entirely separate from the randomness that is used in Exp.

3.6 Proof of Lemma 3.12 (Difference between $\mathbf{Hyb}_0^{\ell+1}$ and $\mathbf{Hyb}_1^{\ell+1}$)

We will conclude Lemma 3.12 from the following technical result that may be of independent interest.

Definition 3.16. *Let $\mathcal{H}$ be a random oracle (with input space $\mathcal{I}$ and output space $\mathcal{O}$). Suppose that $P^{\mathcal{H}}$ is a procedure (without input and returning a single bit) that has access to $\mathcal{H}$, such that P* is guaranteed to never make the same query twice. *Note that we can view P as a randomized function that maps* transcripts *(i.e., a list of previous oracle input/output pairs) to either the next query $x \in \mathcal{I}$ or to an output in $\{0,1\}$.*

Now suppose that the RO $\mathcal{H}$ is fixed. Then we say that the oracle $\widehat{\mathcal{H}}$ (which has input space $\mathcal{I} \times \{0,1\}$ and output space $\mathcal{O}$) extends *the RO $\mathcal{H}$, if $\widehat{\mathcal{H}}(x,1) = \mathcal{H}(x)$ for all $x \in \mathcal{I}$ and $\widehat{\mathcal{H}}(x,0)$ is assigned a uniformly random output $y \in \mathcal{O}$. Given a fixed RO $\mathcal{H}$, we call a procedure $\widehat{P}^{\widehat{\mathcal{H}}}$* oracle-fixing *for P, if for each query (x,b) it makes, the marginal distribution of x is identical to the distribution of query x made by P (both conditioned on the transcript so far); and if the output distributions of P and $\widehat{P}$ (conditioned on the entire transcript) are identical in both cases.*

Intuitively, an oracle-fixing procedure $\widehat{P}$ must be identical to the original procedure P, except that for each query that it makes, it may choose to either use the output of a previously-fixed oracle $\mathcal{H}$ (whose entire function table is public) or to query a fresh RO (whose function table is unknown). We can now prove the following result, showing that despite this capability, when the first oracle is a uniformly random RO $\mathcal{H}$, then with sufficient probability, no query-fixing procedure $\widehat{P}$ (that is given access to a fresh RO $\widehat{\mathcal{H}}$) exists that deviates too much from the original procedure P.

Lemma 3.17. *Suppose that $P^{\mathcal{H}}$ is a procedure (without input and returning a single bit) that is guaranteed to make at most q queries to a random oracle $\mathcal{H}$. Let*

$$p := \Pr_{\mathcal{H}}[P^{\mathcal{H}} \text{ outputs } 0]$$

be the probability of P returning 0 (taken over P's internal randomness as well as the RO $\mathcal{H}$).

Then, the following yields a bound regarding the expectation of output 0 for the optimal choice *of $\widehat{P}$ (where "optimal" refers to probability of output 0):*

$$\mathop{\mathbb{E}}_{\mathcal{H}}\left[\max_{\widehat{P}} \Pr_{\widehat{\mathcal{H}}}\left[\widehat{P}^{\widehat{\mathcal{H}}} \text{ outputs } 0\right]\right] \leq 2^q \cdot p$$

Note that the expectation $\mathbb{E}_{\mathcal{H}}$ is only taken over the choice of RO $\mathcal{H}$, the maximum is taken over all query-fixing procedures $\widehat{P}$ of P (which may depend on the entire function table of $\mathcal{H}$ since it is already fixed), and the inner probability is taken over both P's internal randomness and the random choice of an extension $\widehat{\mathcal{H}}$ of $\mathcal{H}$.

Consequently:

$$\Pr_{\mathcal{H}}\left[\max_{\widehat{P}} \Pr_{\widehat{\mathcal{H}}}\left[\widehat{P}^{\widehat{\mathcal{H}}} \text{ outputs } 0\right] > 2^{q+2} \cdot p\right] < \frac{1}{4}\,.$$

Proof. W.l.o.g. we assume that P makes *exactly* q queries (otherwise, P can be transformed into some equivalent P' that performs some dummy queries whose output is ignored; if the input space $\mathcal{I}$ is not large enough, we can simply make it larger without changing the statement).

We proceed by induction on q. For $q = 0$, the claim is trivial, since P does not make any RO queries, and therefore every "transcript" that determines the output of P will always be empty. Thus, every query-fixing $\widehat{P}$ will be identical to P, i.e.,

$$\Pr_{\widehat{\mathcal{H}}}\left[\widehat{P}^{\widehat{\mathcal{H}}} \text{ outputs } 0\right] = p = 2^q \cdot p\,.$$

Now assume that the claim holds for $q-1$. We show that it also holds for q, so let P be a procedure that makes exactly q queries. Note that P can be rewritten in the following form:

1. P selects (potentially non-deterministically) a query $x^* \in \mathcal{I}$ that it sends to its oracle.
2. P receives an output $y^* \in \mathcal{O}$ from the oracle.
3. P runs P_{x^*,y^*}, where P_{x^*,y^*} is a procedure that has the current transcript (x^*, y^*) hardcoded and makes exactly $q-1$ queries to the RO.
4. P returns the output of P_{x^*,y^*}.

In order to prove the lemma, we need to upper-bound $\mathbb{E}_{\mathcal{H}}[S]$, where S is the following random variable that depends on $\mathcal{H}$:

$$S := \max_{\widehat{P}} \Pr_{\widehat{\mathcal{H}}}\left[\widehat{P}^{\widehat{\mathcal{H}}} \text{ outputs } 0\right] .$$

(Again, $\widehat{P}$ is any query-fixing procedure of P that may depend on the RO $\mathcal{H}$, and the probability is taken over a uniformly random extension $\widehat{\mathcal{H}}$ of $\mathcal{H}$.)

For every query $x \in \mathcal{I}$, we define the random variable S_x identically, except that the probability in the definition of S_x is additionally *conditioned on the first query being* x. We get (where $\Pr[x^* = x]$ denotes the probability that the procedure P chooses x as its first query)

$$\mathop{\mathbb{E}}_{\mathcal{H}}[S] \leq \sum_{x \in \mathcal{I}} \Pr[x^* = x] \cdot \mathop{\mathbb{E}}_{\mathcal{H}}[S_x] , \tag{15}$$

because of $\max_{\widehat{P}} \sum_x v_{\widehat{P},x} \leq \sum_x \max_{\widehat{P}} v_{\widehat{P},x}$ for arbitrary values of $v_{\widehat{P},x}$. In addition, for every input x and every output $y \in \mathcal{O}$, we define the random variable $S_{x,y}$ identically to S, except that the procedure $P_{x,y}$ instead of P is used.

Fix $\mathcal{H}$ and some query x, and let $\widehat{P}$ be the best corresponding query-fixing procedure (i.e., the one for which the term inside the definition of S_x is maximized). Its first query is either $(x, 0)$ or $(x, 1)$. Thus, since $\widehat{\mathcal{H}}(x, 1) = \mathcal{H}(x)$ while $\widehat{\mathcal{H}}(x, 0)$ is chosen uniformly from $\mathcal{O}$, combined with the fact that $P_{x,y}$ is guaranteed to never query x, we have

$$S_x = \max\left(S_{x,\mathcal{H}(x)}, \frac{1}{|\mathcal{O}|}\sum_{y \in \mathcal{O}} S_{x,y}\right) .$$

Hence, the expected value (with $\mathcal{H}$ not being fixed anymore – i.e., $\widehat{P}$ is also a random variable) is

$$\mathop{\mathbb{E}}_{\mathcal{H}}[S_x] = \mathop{\mathbb{E}}_{\mathcal{H}}\left[\max\left(S_{x,\mathcal{H}(x)}, \frac{1}{|\mathcal{O}|}\sum_{y \in \mathcal{O}} S_{x,y}\right)\right] \leq \mathop{\mathbb{E}}_{\mathcal{H}}[S_{x,\mathcal{H}(x)}] + \frac{1}{|\mathcal{O}|}\sum_{y \in \mathcal{O}} \mathop{\mathbb{E}}_{\mathcal{H}}[S_{x,y}] , \tag{16}$$

where we used the fact $\max(R_1, R_2) \leq R_1 + R_2$ for any two real values $R_1, R_2 \geq 0$, together with linearity of expectation.

The term $\mathbb{E}_{\mathcal{H}}[S_{x,\mathcal{H}(x)}]$ is equal to

$$\mathop{\mathbb{E}}_{\mathcal{H}}[S_{x,\mathcal{H}(x)}] = \sum_{y \in \mathcal{O}} \mathop{\mathbb{E}}_{\mathcal{H}}[S_{x,\mathcal{H}(x)} \mid \mathcal{H}(x) = y] \cdot \Pr[\mathcal{H}(x) = y] = \frac{1}{|\mathcal{O}|}\sum_{y \in \mathcal{O}} \mathop{\mathbb{E}}_{\mathcal{H}}[S_{x,y}] . \tag{17}$$

For every input x and every output $y \in \mathcal{O}$, note that by induction applied to procedure $P_{x,y}$, we get

$$\mathop{\mathbb{E}}_{\mathcal{H}}[S_{x,y}] \leq 2^{q-1} \cdot \Pr_{\mathcal{H}}[P_{x,y}^{\mathcal{H}} \text{ outputs } 0] , \tag{18}$$

By combining Equations (15) to (18), we obtain

$$\mathop{\mathbb{E}}_{\mathcal{H}}[S] \leq 2^q \cdot \frac{1}{|\mathcal{O}|} \sum_{\substack{x \in \mathcal{I} \\ y \in \mathcal{O}}} \Pr[x^* = x] \cdot \Pr_{\mathcal{H}}[P_{x,y}^{\mathcal{H}} \text{ outputs } 0]$$

Note that the term on the right is equal to $2^q \cdot p$ by definition of P (using the fact that $P_{x,y}$ will never query x and therefore it does not matter if the probability on the right is conditioned on $\mathcal{H}(x) = y$ or not). $\square$

We are now ready to prove Lemma 3.12, which we recall here:

Lemma 3.12. *In the experiment* Exp*, we say that the event* Bad_1 *holds whenever there exists a valid vector space* $L'_{\mathsf{all}} \subseteq \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n_{\mathfrak{g}_{\ell+1}}} \rangle$ *with* $L'_{\mathsf{all}} \supseteq L_{\ell+1}$ *and a valid set* $\mathcal{S}' \supseteq \mathcal{S}_{\ell+1}$*, s.t.*

$$\begin{aligned} &\Pr\left[\mathrm{HYB}_1^{\ell+1}(L'_{\mathsf{all}}, \mathcal{S}') : (b = 0 \vee \mathsf{BadVerifyQuery}) \wedge \overline{\mathsf{InconsistentRO}}\right] \\ &> 2^{q+2} \cdot \Pr\left[\mathrm{HYB}_0^{\ell+1} : b = 0 \vee \mathsf{BadVerifyQuery}\right] . \end{aligned}$$

We have $\Pr[\mathsf{Exp} : \mathsf{Bad}_1] \leq \frac{1}{4}$.

Proof. Consider a fixed execution of the experiment Exp. We define the following procedure P that has oracle access to some $\mathcal{H}^*$ whose input space is $\mathcal{I} := (\mathbb{Z}_p^{n^{\mathsf{H}}} \times \{0,1\}^*) \setminus (\mathcal{R} \cup \mathcal{Q}_{\ell+1}^{\mathsf{User}_1} \cup \mathcal{Q}_{\ell+1}^{\mathsf{Sign}})$ (i.e., the usual input space of $\mathcal{H}$, except that all inputs from $\mathcal{R} \cup \mathcal{Q}_{\ell+1}^{\mathsf{User}_1} \cup \mathcal{Q}_{\ell+1}^{\mathsf{Sign}}$ are removed) and whose output space is $\{0,1\}^r$:

1. Run $\widetilde{\sigma} \leftarrow \widetilde{\mathsf{User}}_2^{\mathcal{O}_{\mathsf{eq}}[\mathfrak{g}_{\ell+1}],\mathcal{O}_{\mathsf{H}}}(\widetilde{\mathsf{st}}_{\ell+1}, \widetilde{\mathsf{msgS}}_{\ell+1})$ and $b \leftarrow \widetilde{\mathsf{Verify}}^{\mathcal{O}_{\mathsf{eq}}[\mathfrak{g}_{\ell+1}],\mathcal{O}_{\mathsf{H}}}(\widetilde{\mathsf{vk}}, m_{\ell+1}, \widetilde{\sigma})$, where $\mathcal{O}_{\mathsf{H}}(M, x)$ is implemented as follows:
   - If a query (M', x) with $M\mathfrak{g}_{\ell+1} = M'\mathfrak{g}_{\ell+1}$ was made before, return the same output as before.
   - Otherwise, if $(M\mathfrak{g}_{\ell+1}, x) \in \mathcal{R} \cup \mathcal{Q}_{\ell+1}^{\mathsf{User}_1} \cup \mathcal{Q}_{\ell+1}^{\mathsf{Sign}}$, then return $\mathcal{H}(M\mathfrak{g}_{\ell+1}, x)$.
   - Otherwise, query $\mathcal{H}^*(M\mathfrak{g}_{\ell+1}, x)$ and return its output.
2. Output 1 iff $b = 1 \wedge \overline{\mathsf{BadVerifyQuery}}$.

Note that the procedure P (given a fresh RO $\mathcal{H}^*$) is identical to the hybrid $\mathrm{HYB}_0^{\ell+1}$, because both of them answer queries (M, x) using either $\mathcal{H}$ (if the query matches an input in $\mathcal{R} \cup \mathcal{Q}_{\ell+1}^{\mathsf{User}_1} \cup \mathcal{Q}_{\ell+1}^{\mathsf{Sign}}$) or else uniformly random. Thus:

$$\Pr_{\mathcal{H}^*}\left[P^{\mathcal{H}^*} \text{ outputs } 0\right] = \Pr\left[\mathrm{HYB}_0^{\ell+1} : b = 0 \vee \mathsf{BadVerifyQuery}\right] \tag{19}$$

Now fix some RO $\mathcal{H}^*$, as well as any valid L'_{all} and any set of RO queries $\mathcal{S}' \supseteq \mathcal{S}_{\ell+1}$ that is valid w.r.t. $\mathcal{H}$ (on inputs $\mathcal{R} \cup \mathcal{Q}_{\ell+1}^{\mathsf{User}_1} \cup \mathcal{Q}_{\ell+1}^{\mathsf{Sign}}$) and $\mathcal{H}^*$ (on all other inputs). Given these, we define a query-fixing procedure $\widehat{P}_{L'_{\mathsf{all}},\mathcal{S}'}$ for P (see Definition 3.16) as follows:

1. Run $\widetilde{\sigma} \leftarrow \widetilde{\mathsf{User}}_2^{\mathcal{O}_{\mathsf{eq}}[\mathfrak{g}_{\ell+1}],\mathcal{O}_{\mathsf{H}}}(\widetilde{\mathsf{st}}_{\ell+1}, \widetilde{\mathsf{msgS}}_{\ell+1})$ and $b \leftarrow \widetilde{\mathsf{Verify}}^{\mathcal{O}_{\mathsf{eq}}[\mathfrak{g}_{\ell+1}],\mathcal{O}_{\mathsf{H}}}(\widetilde{\mathsf{vk}}, m_{\ell+1}, \widetilde{\sigma})$, where $\mathcal{O}_{\mathsf{H}}(M, x)$ is implemented as follows:
   - If a query (M', x) with $M\mathfrak{g}_{\ell+1} = M'\mathfrak{g}_{\ell+1}$ was made before, return the same output as before.
   - Otherwise, if $(M\mathfrak{g}_{\ell+1}, x) \in \mathcal{R} \cup \mathcal{Q}^{\mathsf{User}_1} \cup \mathcal{Q}^{\mathsf{Sign}}$, then return $\mathcal{H}(M\mathfrak{g}_{\ell+1}, x)$.
   - Otherwise, if $\mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L'_{\mathsf{all}}, \mathcal{S}'](M, x) \neq \bot$, then query $((M\mathfrak{g}_{\ell+1}, x), 1)$ to $\widehat{\mathcal{H}}^*$ and return its output.
   - Otherwise, query $((M\mathfrak{g}_{\ell+1}, x), 0)$ to $\widehat{\mathcal{H}}^*$ and return its output.
2. Output 1 iff $b = 1 \wedge \overline{\mathsf{BadVerifyQuery}}$.

Note that the query-fixing procedure $\widehat{P}_{L'_{\mathsf{all}},\mathcal{S}'}$ (given a randomly chosen extension $\widehat{\mathcal{H}}^*$ of $\mathcal{H}^*$) is *almost* identical to the hybrid $\mathrm{HYB}_1^{\ell+1}(L'_{\mathsf{all}}, \mathcal{S}')$. The only difference is that $\widehat{P}_{L'_{\mathsf{all}},\mathcal{S}'}$ never returns two different answers for two queries $(M, x), (M', x)$ with $M\mathfrak{g}_{\ell+1} = M'\mathfrak{g}_{\ell+1}$ while $\mathrm{HYB}_1^{\ell+1}$ might (because for the latter, it is possible e.g. that $\mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L'_{\mathsf{all}}, \mathcal{S}']$ returns $\bot$ when given (M, x) but not given (M', x)).

Whenever this happens, the event $\mathsf{InconsistentRO}$ must apply, and hence we can still conclude the relation

$$\Pr_{\widehat{\mathcal{H}}^*}\left[\widehat{P}^{\widehat{\mathcal{H}}^*}_{L'_{\mathsf{all}},\mathcal{S}'} \text{ outputs } 0\right] \geq \Pr\left[\mathrm{HYB}_1^{\ell+1}(L'_{\mathsf{all}},\mathcal{S}') : (b=0 \vee \mathsf{BadVerifyQuery}) \wedge \overline{\mathsf{InconsistentRO}}\right] . \quad (20)$$

Given the fixed execution of Exp, define $p := \Pr_{\mathcal{H}^*}\left[P^{\mathcal{H}^*} \text{ outputs } 0\right]$ as the probability of P returning 0 for a uniformly random $\mathcal{H}^*$. We can now apply Lemma 3.17 to procedure P, which shows

$$\Pr_{\mathcal{H}^*}\left[\exists \text{ valid } L'_{\mathsf{all}},\mathcal{S}' \text{ with } \Pr_{\widehat{\mathcal{H}}^*}\left[\widehat{P}^{\widehat{\mathcal{H}}^*}_{L'_{\mathsf{all}},\mathcal{S}'} \text{ outputs } 0\right] > 2^{q+2}\cdot p\right] < \frac{1}{4} , \quad (21)$$

where "validity" of $\mathcal{S}'$ is defined w.r.t. $\mathcal{H}$ (on inputs $\mathcal{R}\cup\mathcal{Q}^{\mathsf{User}_1}_{\ell+1}\cup\mathcal{Q}^{\mathsf{Sign}}_{\ell+1}$) and $\mathcal{H}^*$ (on all other inputs), and the inner probability is taken over a uniformly random extension $\widehat{\mathcal{H}}^*$ of $\mathcal{H}^*$.

Note that while Exp may make RO queries to $\mathcal{H}$ that are outside of $\mathcal{R}\cup\mathcal{Q}^{\mathsf{User}_1}_{\ell+1}\cup\mathcal{Q}^{\mathsf{Sign}}_{\ell+1}$, none of their outputs affect any of the variables used inside P. In fact, instead of running the entire Exp, we could equivalently just run its first 3 steps as well as InitSession and ObtainMsgS of the $(\ell+1)$-th iteration of step 4 (see Figure 8).[18] Therefore, we can view the fresh RO $\mathcal{H}^*$ (whose input space encompasses exactly those queries that are *not* in $\mathcal{R}\cup\mathcal{Q}^{\mathsf{User}_1}_{\ell+1}\cup\mathcal{Q}^{\mathsf{Sign}}_{\ell+1}$) as the "remainder" of the RO $\mathcal{H}$ that has not been queried yet by Exp and therefore is information-theoretically hidden in P.

Given a fixed execution of Exp, let Bad'_1 be true iff the event inside the probability in (21) is true for the unique RO $\mathcal{H}^*$ with $\mathcal{H}^*(\mathfrak{h},x)=\mathcal{H}(\mathfrak{h},x)$ for all $(\mathfrak{h},x)\in\mathcal{I}$. Then we have $\Pr[\mathsf{Exp}:\mathsf{Bad}'_1]<\frac{1}{4}$ by the discussion above and (21). Furthermore, combining Equations (19) and (20), we obtain the implication $\mathsf{Bad}_1 \Rightarrow \mathsf{Bad}'_1$. Thus, $\Pr[\mathsf{Exp}:\mathsf{Bad}_1]<\frac{1}{4}$. □

3.7 Proof of Lemma 3.14 (Bounding $\mathsf{MissingKnowledge}_{\ell+1}$)

We recall the lemma:

Lemma 3.14. *The following holds for sufficiently large λ:*

$$\Pr[\mathsf{Exp}:\mathsf{MissingKnowledge}_{\ell+1}] < \frac{1}{4} .$$

Proof. In the following, we abuse notation to write $\mathcal{S}\cap\mathcal{R}$ to denote the set

$$\{(\mathfrak{h},x) \mid \exists (M,x,y)\in\mathcal{S} \text{ with } M\mathfrak{g}=\mathfrak{h}\}$$

of RO inputs in $\mathcal{R}$ that have a matching triple in $\mathcal{S}$. Note that $|\mathcal{S}\cap\mathcal{R}|\leq|\mathcal{R}|$ for any $\mathcal{S}$.

Given a set $\mathcal{S}$, we define a modified set $\widehat{\mathcal{S}}\subseteq\mathcal{S}$, which for every unique pair $(\mathfrak{h},x)$ may contain *at most one* triple (M,x,y) with $M\mathfrak{g}=\mathfrak{h}$. If there are multiple triples mapping to the same $(\mathfrak{h},x)$, it does not matter which ones of them are deleted as long as this is done consistently (for example, we could use the convention we only keep the lexicographically smallest triple). Now we also define $\widehat{V}_i$ for each iteration i in the same way as V_i, except that in its definition we replace $\mathcal{S}_i$ by $\widehat{\mathcal{S}}_i$.

Clearly, we always have $\mathcal{S}_i\subseteq\mathcal{S}^*_i\subseteq\mathcal{S}_{i+1}$ and $L_i\cap\widehat{V}_i\subseteq L^*_i\cap\widehat{V}_i\subseteq L_{i+1}\cap\widehat{V_{i+1}}$ (we only ever *add* new elements to any of these sets). Furthermore, we have $\dim\widehat{V}_i\leq 1+n^{\mathsf{vk}}+|\mathcal{R}|\cdot n^{\mathsf{H}}$ for every i because there cannot be any two different $(M,x,y),(M',x,y')\in\mathcal{S}$ with $M\cdot\mathfrak{g}=M'\cdot\mathfrak{g}$. Thus, denoting by

$$\mathcal{I} := \{i \mid \mathcal{S}_i\cap\mathcal{R}\subsetneq\mathcal{S}^*_i\cap\mathcal{R} \ \vee\ L_i\cap\widehat{V}_i\subsetneq L^*_i\cap\widehat{V}_i\}$$

[18] Strictly speaking, we would also need to increment $n^{\mathfrak{g}}_i := n^{\mathfrak{g}}_{i-1}+n^{\mathsf{msgS}}$ in each of the remaining iterations i. However, for running the procedure P, only the group elements $\mathfrak{g}_{\ell+1}[1\ldots n^{\mathsf{vk}}+1]$ and $\mathfrak{g}_{\ell+1}[n^{\mathfrak{g}}_\ell+1\ldots n^{\mathfrak{g}}_{\ell+1}]$ are relevant and therefore all other group elements can be set to e.g. $\mathfrak{o}$.

those iterations in which either $\mathcal{S}_i \cap \mathcal{R}$ or $L_i \cap \widehat{V}_i$ grows, then we have $|\mathcal{I}| \leq 1 + n^{\mathsf{vk}} + |\mathcal{R}| \cdot (n^{\mathsf{H}} + 1) \leq \frac{1}{4} \cdot \ell_{\mathsf{max}}$.

Later we will prove that, taken over the experiment Exp, for every $i \in [\ell_{\mathsf{max}}]$,

$$\Pr[\mathsf{MissingKnowledge}_i \Rightarrow i \in \mathcal{I}] \geq 1 - \mathrm{negl}(\lambda) \; . \tag{22}$$

Thus, denoting by $\mathcal{I}' := \{i \mid \mathsf{MissingKnowledge}_i\}$ those iterations where $\mathsf{MissingKnowledge}_i$ holds, by union-bound we have

$$\Pr\left[|\mathcal{I}'| > \frac{1}{4} \cdot \ell_{\mathsf{max}}\right] \leq \underbrace{\Pr\left[|\mathcal{I}| > \frac{1}{4} \cdot \ell_{\mathsf{max}}\right]}_{=0} + \mathrm{negl}(\lambda) < \mathrm{negl}(\lambda) \; .$$

Then, because ℓ is randomly picked from $\{0, \ldots, \ell_{\mathsf{max}}\}$ (and it is not used anywhere within Exp), the probability of $\mathsf{MissingKnowledge}_{\ell+1}$ is

$$\begin{aligned} \Pr[\ell + 1 \in \mathcal{I}'] &\leq \Pr\left[|\mathcal{I}'| > \frac{1}{4} \cdot \ell_{\mathsf{max}}\right] + \Pr\left[\ell + 1 \in |\mathcal{I}'| \;\middle|\; |\mathcal{I}'| \leq \frac{1}{4} \cdot \ell_{\mathsf{max}}\right] \\ &\leq \mathrm{negl}(\lambda) + \frac{1}{4} \cdot \frac{\ell_{\mathsf{max}}}{\ell_{\mathsf{max}} + 1} \leq \frac{1}{4} \end{aligned}$$

for sufficiently large λ.

It remains to prove (22). To this end, fix an execution of Exp, and suppose that $\mathsf{MissingKnowledge}_i$ holds for some i. By Definition 3.13, this means that there exists a valid $L'_{\mathsf{all}} \subseteq \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}_{i+1}} \rangle$ with $L'_{\mathsf{all}} \supseteq L_i$ as well as a valid $\mathcal{S}' \supseteq \mathcal{S}_i$ for which (12) holds. For all of these fixed values, we study the experiment $\textsc{GatherKnowledgeRand}^{\mathcal{H}}(\mathfrak{g}_i, \mathcal{S}_i, \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}_i, \widetilde{\mathsf{msgS}}_i, m_i)$, for which we say that the event $\mathsf{CorrectGuesses}$ is true if for every query (M, x) made by either $\widetilde{\mathsf{User}}_2$ or $\widetilde{\mathsf{Verify}}$, the random bit b chosen within $\mathsf{Randomize}\mathcal{O}_{\mathsf{H}}[\mathcal{S}_{\mathsf{rand}}]$ is 1 iff $\mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L'_{\mathsf{all}}, \mathcal{S}'](M, x) = \bot$.

Conditioned on $\mathsf{CorrectGuesses}$, we have

$$\begin{aligned} &\Pr\left[\textsc{GatherKnowledgeRand}^{\mathcal{H}}(\mathfrak{g}_i, \mathcal{S}_i, \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}_i, \widetilde{\mathsf{msgS}}_i, m_i) : \begin{array}{l}\mathsf{LearnedRelation} \\ \vee\ \mathsf{LearnedQuery}\end{array} \;\middle|\; \mathsf{CorrectGuesses}\right] \\ = \; &\Pr\left[\textsc{Hyb}_1^i(L'_{\mathsf{all}}, \mathcal{S}') : \begin{array}{l}\mathsf{LearnedRelation} \\ \vee\ \mathsf{LearnedQuery}\end{array}\right] \; . \end{aligned}$$

To see this, note that the oracles that $\textsc{GatherKnowledgeRand}$ provides to $\widetilde{\mathsf{User}}_2$ and $\widetilde{\mathsf{Verify}}$ behave identically to those in $\textsc{Hyb}_1$, because $\mathcal{S}'$ is valid and thus whenever $\mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L'_{\mathsf{all}}, \mathcal{S}']$ outputs something other than $\bot$, this output always matches that of the real RO $\mathcal{H}$, i.e., exactly the same as what $\mathsf{Randomize}\mathcal{O}_{\mathsf{H}}[\mathcal{S}_{\mathsf{rand}}]$ would return due to $b = 0$. Moreover, all other queries (not captured by $\mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L'_{\mathsf{all}}, \mathcal{S}']$) will receive a uniformly random output (either due the use of $\mathsf{Random}\mathcal{O}_{\mathsf{H}}$ when $b = 1$ in the first case, or due to the use of a fresh RO $\mathcal{H}'$ in the latter case). If such a query is made a second time (and still not captured by $\mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L'_{\mathsf{all}}, \mathcal{S}']$), then the same output will be given as the first time: in the former case, note that $\mathsf{Randomize}\mathcal{O}_{\mathsf{H}}$ ensures this by looking up the query in $\mathcal{S}_{\mathsf{rand}}$, and in the latter case, $\mathcal{H}'$ is always consistent with repeated queries by definition.

Since the probability of $\mathsf{CorrectGuesses}$ is clearly $\geq \frac{1}{2^q}$, the following event holds in Exp conditioned on $\mathsf{MissingKnowledge}_i$:

$$\Pr\left[\textsc{GatherKnowledgeRand}^{\mathcal{H}}(\mathfrak{g}_i, \mathcal{S}_i, \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}_i, \widetilde{\mathsf{msgS}}_i, m_i) : \begin{array}{l}\mathsf{LearnedRelation} \\ \vee\ \mathsf{LearnedQuery}\end{array}\right] \geq \frac{1}{4 \cdot 2^q}$$

Note that our OMUF adversary repeats $\textsc{GatherKnowledgeRand}$ for $k_{\mathsf{rand}} = \lambda \cdot 4 \cdot 2^q$ independent times. Thus, given the above, $\mathsf{LearnedRelation} \vee \mathsf{LearnedQuery}$ will be true for at least one of these k_{rand} iterations with probability

$$\geq 1 - \left(1 - \frac{1}{4 \cdot 2^q}\right)^{\lambda \cdot 4 \cdot 2^q} \geq 1 - \frac{1}{e^\lambda} > 1 - \mathrm{negl}(\lambda) \; .$$

Furthermore, if for at least one of Exp's iterations $j \in [k_{\mathsf{rand}}]$ of GatherKnowledgeRand, one of the two events is true, then we get the following:

- $\mathsf{LearnedQuery}$: There is some $(M, x, y) \in \mathcal{S}^{(j)}_{\mathsf{new}}$ with $(M \cdot \mathfrak{g}, x) \in \mathcal{R}$, but there is no other $(M', x, y') \in \mathcal{S}_i$ with $M \cdot \mathfrak{g} = M' \cdot \mathfrak{g}$. In this case, we will have $(M, x, y) \in \mathcal{S}_{i,j} \subseteq \mathcal{S}^*_i$ and therefore $\mathcal{S}_i \cap \mathcal{R} \subsetneq \mathcal{S}^*_i \cap \mathcal{R}$.
- $\mathsf{LearnedRelation}$: $L^{(j)}_{\mathsf{new}} \cap V_i \not\subseteq L_i$, i.e., there exists a vector $\mathbf{v} \in L^{(j)}_{\mathsf{new}} \cap V_i$ with $\mathbf{v} \notin L_i$. Due to $L^{(j)}_{\mathsf{new}} \subseteq L^*_i$ (by definition of L^*_i), this means $\mathbf{v} \in L^*_i \cap V_i$.
  Recall that the only difference between V_i and $\widehat{V}_i$ is that the first one sums up all $\mathrm{rowsp}(M)$ for $(M, x, y) \in \mathcal{S}$ (s.t. $(M\mathfrak{g}, x) \in \mathcal{R}$), while $\widehat{V}_i$ only takes triples in $(M, x, y) \in \widehat{\mathcal{S}}$. However, whenever $\mathcal{A}$ calls the RO on two queries (M, x) and (M', x) with $M\mathfrak{g} = M'\mathfrak{g}$, then (due to $\mathsf{LearnRel} : (L_{\mathsf{new}}, \mathcal{S})$ being activated) it adds $\mathrm{rowsp}(M - M')$ to the set of linear relations. Hence, for any $\mathbf{v} \in V_i$, there exists some $\mathbf{v}' \in L_i$ with $\mathbf{v} - \mathbf{v}' \in \widehat{V}_i$.
  Hence, we have $\mathbf{v} - \mathbf{v}' \in L^*_i \cap \widehat{V}_i$ and simultaneously $\mathbf{v} - \mathbf{v}' \notin L_i \cap \widehat{V}_i$ (due to $\mathbf{v} \notin L_i$ and $\mathbf{v}' \in L_i$). Thus, we can conclude $L_i \cap \widehat{V}_i \subsetneq L^*_i \cap \widehat{V}_i$.

This proves (22). □

3.8 Proof of Lemma 3.15 (Difference between $\mathrm{Hyb}_1^{\ell+1}$ and SimForge)

We recall the lemma:

Lemma 3.15. *Consider an execution of the experiment* Exp *for which the event* $\overline{\mathsf{MissingKnowledge}_{\ell+1}}$ *holds. Then, there must exist a tuple* $(L_{\mathsf{forgery}}, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}})$ *with* $|\mathcal{S}_{\mathsf{fake}}| \leq q_{\mathsf{Sign}}$ *and*

$$L_{\mathsf{forgery}}, L_{\mathsf{fake}} \subseteq \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}_{\ell+1}} \rangle \quad \text{and} \quad L_{\mathsf{forgery}} \cap \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}_{\ell}} \rangle \subseteq L_{\ell+1}$$

s.t. for every valid $L' \subseteq \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}_{\ell}} \rangle$ *with* $L' \supseteq L_{\ell+1}$ *and for every valid* $\mathcal{S}' \supseteq \mathcal{S}_{\ell+1}$*, we have*

$$\begin{aligned} &\Pr[\textsc{SimForge}(L' + L_{\mathsf{forgery}}, \mathcal{S}', \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}_{\ell+1}, m_{\ell+1}, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}}) \text{ outputs } 1] \\ \geq\ &\Pr[\textsc{Hyb}^{\ell+1}_1(L' + L_{\mathsf{forgery}}, \mathcal{S}') : (b = 1 \wedge \overline{\mathsf{BadVerifyQuery}}) \vee \mathsf{InconsistentRO}] - \frac{1}{4}\,. \end{aligned}$$

Proof. We start by constructing the tuple $(L_{\mathsf{forgery}}, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}})$, in which we essentially imitates the "real" signer message $\mathsf{msgS}_{\ell+1}$. (Recall that $\mathsf{msgS}_{\ell+1}$ is present in experiment Exp, even though it would not be available to the adversary $\mathcal{A}$ in the $\mathbf{omuf}_{\mathsf{BS},\mathcal{A}}$ game. Of course the adversary $\mathcal{A}$ may be unable to come up with this exact tuple, but recall that it suffices for us to prove *existence* of a tuple for which the adversary may successfully simulate a forgery session.)

- $L_{\mathsf{forgery}} := \{\mathbf{v} \mid \mathbf{v}^{\mathsf{T}} \mathfrak{g}_{\ell+1} = \mathfrak{o}\} \cap (L_{\ell+1} + \langle \mathbf{e}_{n^{\mathfrak{g}}_{\ell}+1}, \ldots, \mathbf{e}_{n^{\mathfrak{g}}_{\ell+1}} \rangle)$.
  (Intuitively, this L_{forgery} *includes all linear constraints about the "real" signer-message elements* $\mathfrak{g}_{\ell+1}$*, as long as they are explainable using the existing knowledge (i.e., using* $L_{\ell+1} + \langle \mathbf{e}_{n^{\mathfrak{g}}_{\ell}+1}, \ldots, \mathbf{e}_{n^{\mathfrak{g}}_{\ell+1}} \rangle$*).)*
  Note that $L_{\mathsf{forgery}} \cap \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}_{\ell}} \rangle \subseteq (L_{\ell+1} + \langle \mathbf{e}_{n^{\mathfrak{g}}_{\ell}+1}, \ldots, \mathbf{e}_{n^{\mathfrak{g}}_{\ell+1}} \rangle) \cap \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}_{\ell}} \rangle = L_{\ell+1}$, and therefore the required condition $L_{\mathsf{forgery}} \cap \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}_{\ell}} \rangle \subseteq L_{\ell+1}$ is fulfilled.
- $s^{\mathsf{msgS}} := s^{\mathsf{msgS}}_{\ell+1}$.
  (We choose exactly the same bitstring as in the "real" signer message.)
- $L_{\mathsf{fake}} := \{\mathbf{v} \mid \mathbf{v}^{\mathsf{T}} \mathfrak{g}_{\ell+1} = \mathfrak{o}\} \cap \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}_{\ell+1}} \rangle$
  (Note that this is essentially the vector space of all *linear constraints, which will be sufficient for the simulation to recognize all queries in* $\mathcal{S}_{\mathsf{fake}}$ *correctly, regardless of their representation.)*
- $\mathcal{S}_{\mathsf{fake}} := \{(M, x, \mathcal{H}(\mathfrak{h}, x)) \mid (\mathfrak{h}, x) \in \mathcal{Q}^{\mathsf{Sign}}_{\ell+1}\}$, where M is an arbitrarily chosen matrix M with $M\mathfrak{g} = \mathfrak{h}$ (for example, we may choose the first column of M to be equal to all dlog's of $\mathfrak{h}$ to base $\mathfrak{1}$, and everything else is 0).
  (This is exactly the set of all *queries made by the "real" signer, except that we write its triples in matrix form so that it is compatible with* SimForge*.)*

Now consider any valid vector space L' with $L_{\ell+1} \subseteq L' \subseteq \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n_\ell^{\mathfrak{g}}} \rangle$ as stated in the lemma, and consider any valid set $\mathcal{S}' \supseteq \mathcal{S}_{\ell+1}$ of RO queries. In the following, the use the shorthand notation $L'_{\mathsf{all}} := L' + L_{\mathsf{forgery}}$.

We will show that, unless one of the three events $\mathsf{LearnedRelation}$, $\mathsf{LearnedQuery}$, or $\mathsf{BadVerifyQuery}$ holds, the distribution of b induced by

$$\textsc{Hyb}_1^{\ell+1}(L'_{\mathsf{all}}, \mathcal{S}')$$

is identical to the output distribution of

$$\textsc{SimForge}(L'_{\mathsf{all}}, \mathcal{S}', \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}_{\ell+1}, m_{\ell+1}, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}}) \ .$$

In addition, we will prove that in the former, $\mathsf{LearnedRelation}$ is implied by $\mathsf{InconsistentRO}$. Combining all of this, the probability of $\textsc{SimForge}$ to output 1 is at least

$$\begin{aligned} & \Pr[b = 1 \wedge \overline{\mathsf{BadVerifyQuery}} \wedge \overline{\mathsf{LearnedRelation}} \wedge \overline{\mathsf{LearnedQuery}}] \\ = \ & \Pr[((b = 1 \wedge \overline{\mathsf{BadVerifyQuery}}) \vee \mathsf{InconsistentRO}) \wedge \overline{\mathsf{LearnedRelation}} \wedge \overline{\mathsf{LearnedQuery}}] \\ \geq \ & \Pr[(b = 1 \wedge \overline{\mathsf{BadVerifyQuery}}) \vee \mathsf{InconsistentRO}] - \frac{1}{4} \ , \end{aligned}$$

where all probabilities are taken over running $\textsc{Hyb}_1^{\ell+1}(L'_{\mathsf{all}}, \mathcal{S}')$. Thus, we get exactly the lemma's statement as required. Note that the equality follows from the implication $\mathsf{InconsistentRO} \Rightarrow \mathsf{LearnedRelation}$, and the inequality follows from the assumption $\overline{\mathsf{MissingKnowledge}_{\ell+1}}$ (which bounds $\Pr[\mathsf{LearnedRelation} \vee \mathsf{LearnedQuery}] \leq \frac{1}{4}$).

Note that, by definition, the only difference between $\textsc{SimForge}$ and $\textsc{Hyb}_1$ are the oracles ("group equality" and RO) that they provide to the executions of $\widetilde{\mathsf{User}_2}$ and $\widetilde{\mathsf{Verify}}$. We will prove that either those oracles behave identically, or one of the events $\mathsf{LearnedRelation}$, $\mathsf{LearnedQuery}$, or $\mathsf{BadVerifyQuery}$ is true.

First recall that the only group elements that the procedure User_1 takes as input are those in vk, meaning that their representations T^{vk} fulfill $\mathrm{rowsp}(T^{\mathsf{vk}}) \subseteq \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathsf{vk}}+1} \rangle \subseteq V_{\ell+1}$. Because User_1 can compute group elements only as linear combinations of these vectors, its output fulfills $T^{\mathsf{st}}_{\ell+1} \subseteq V_{\ell+1}$.

Similarly, we have $\mathrm{rowsp}(T^{\mathsf{msgS}}_{\ell+1}) = \langle \mathbf{e}_{n_\ell^{\mathfrak{g}}+1}, \ldots, \mathbf{e}_{n_{\ell+1}^{\mathfrak{g}}} \rangle$. Because the input to User_2 consists only of $\widetilde{\mathsf{st}}_{\ell+1}$ $\widetilde{\mathsf{msgS}}_{\ell+1}$, its output fulfills $T^{\mathsf{sig}}_{\ell+1} \subseteq V_{\ell+1} + \langle \mathbf{e}_{n_\ell^{\mathfrak{g}}+1}, \ldots, \mathbf{e}_{n_{\ell+1}^{\mathfrak{g}}} \rangle$.

Therefore, both $\widetilde{\mathsf{User}_2}$ and $\widetilde{\mathsf{Verify}}$ only ever invoke their group equality oracle on vectors $(\mathbf{t}, \mathbf{t}')$ with $\mathbf{t}, \mathbf{t}' \in V_{\ell+1} + \langle \mathbf{e}_{n_\ell^{\mathfrak{g}}+1}, \ldots, \mathbf{e}_{n_{\ell+1}^{\mathfrak{g}}} \rangle$, and they only ever invoke their RO on an input (M, x) with $\mathrm{rowsp}(M) \subseteq V_{\ell+1} + \langle \mathbf{e}_{n_\ell^{\mathfrak{g}}+1}, \ldots, \mathbf{e}_{n_{\ell+1}^{\mathfrak{g}}} \rangle$.

- First, we prove that the group equality oracles

  $$\mathsf{Sim}\mathcal{O}_{\mathsf{eq}}[L'_{\mathsf{all}}] \quad \text{and} \quad \mathcal{O}_{\mathsf{eq}}[\mathfrak{g}_{\ell+1}]$$

  behave identically (the left-hand side is what is used in $\textsc{SimForge}$, and the right-hand side in $\textsc{Hyb}_1$). Consider a query $(\mathbf{t}, \mathbf{t}')$. If $\mathbf{t} - \mathbf{t}' \in L'_{\mathsf{all}}$ (i.e., the left-hand side returns 1), then we clearly also have $(\mathbf{t} - \mathbf{t}') \cdot \mathfrak{g}_{\ell+1} = \mathfrak{o}$ by validity of L'_{all} (and therefore the right-hand side returns 1). Vice versa, if $(\mathbf{t} - \mathbf{t}') \cdot \mathfrak{g}_{\ell+1} = \mathfrak{o}$, then the relation $\mathbf{t} - \mathbf{t}'$ is added to L_{new}. By the above, $\mathbf{t} - \mathbf{t}' \in V_{\ell+1} + \langle \mathbf{e}_{n_\ell^{\mathfrak{g}}+1}, \ldots, \mathbf{e}_{n_{\ell+1}^{\mathfrak{g}}} \rangle$, and unless $\mathsf{LearnedRelation}$ holds, note that we also have $L_{\mathsf{new}} \cap V_{\ell+1} \subseteq L_{\ell+1}$. Thus, $\mathbf{t} - \mathbf{t}' \in L_{\ell+1} + \langle \mathbf{e}_{n_\ell^{\mathfrak{g}}+1}, \ldots, \mathbf{e}_{n_{\ell+1}^{\mathfrak{g}}} \rangle$. By definition of L_{forgery}, this implies $\mathbf{t} - \mathbf{t}' \in L_{\mathsf{forgery}} \subseteq L'_{\mathsf{all}}$.
- Now we prove that the ROs behave identically. Recall that $\textsc{SimForge}$ supplies

  $$\begin{aligned} \mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}}] \prec \mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L'_{\mathsf{all}}, \mathcal{S}' \cup \mathcal{S}_{\mathsf{rand}}] \prec \mathsf{Random}\mathcal{O}_{\mathsf{H}} \quad \text{to} \quad \widetilde{\mathsf{User}_2} \quad \text{and} \\ \mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L'_{\mathsf{all}}, \mathcal{S}' \cup \mathcal{S}_{\mathsf{rand}}] \prec \mathsf{Random}\mathcal{O}_{\mathsf{H}} \quad \text{to} \quad \widetilde{\mathsf{Verify}} \end{aligned}$$

(while $\mathsf{Random}\mathcal{O}_{\mathsf{H}}$ always stores all queries it answers in the initially empty set $\mathcal{S}_{\mathsf{rand}}$), and the procedure $\textsc{Hyb}_1$ supplies

$$\mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L'_{\mathsf{all}}, \mathcal{S}'] \prec \mathcal{O}_{\mathsf{H}}^{\mathcal{H}'}[\mathfrak{g}_{\ell+1}] \quad \text{to both } \widetilde{\mathsf{User}}_2 \text{ and } \widetilde{\mathsf{Verify}} ,$$

which RO $\mathcal{H}'$ being sampled as a fresh RO, except that $\forall(\mathfrak{h}, x) \in \mathcal{R} \cup \mathcal{Q}_{\ell+1}^{\mathsf{User}_1} \cup \mathcal{Q}_{\ell+1}^{\mathsf{Sign}} : \mathcal{H}'(\mathfrak{h}, x) = \mathcal{H}(\mathfrak{h}, x)$. In the following, we will treat $\mathcal{H}'$ as a *lazy* RO that samples the output of a query $(\mathfrak{h}, x) \notin \mathcal{R} \cup \mathcal{Q}_{\ell+1}^{\mathsf{User}_1} \cup \mathcal{Q}_{\ell+1}^{\mathsf{Sign}}$ whenever it is made for the first time.
For convenience, we first unroll the definitions of the oracles above, while simultaneously plugging in our definitions of L_{fake} and $\mathcal{S}_{\mathsf{fake}}$ (which are chosen in such a way that $\mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}}]$ returns $\mathcal{H}(M\mathfrak{g}_{\ell+1}, x)$ iff $(M\mathfrak{g}_{\ell+1}, x) \in \mathcal{Q}_{\ell+1}^{\mathsf{Sign}}$, and $\perp$ otherwise). In particular, given a query (M, x), the oracle provided by $\textsc{SimForge}$ to $\widetilde{\mathsf{User}}_2$ is stated on the left-hand side (for the oracle provided to $\widetilde{\mathsf{Verify}}$, simply disregard case 1), and the oracle provided by $\textsc{Hyb}_1$ to both $\widetilde{\mathsf{User}}_2$ and $\widetilde{\mathsf{Verify}}$ is stated on the right-hand side:

	$\underline{\textsc{SimForge}}$	$\underline{\textsc{Hyb}_1}$
1:	**if** $(M\mathfrak{g}_{\ell+1}, x) \in \mathcal{Q}_{\ell+1}^{\mathsf{Sign}}$ **return** $\mathcal{H}(M\mathfrak{g}_{\ell+1}, x)$	
2:	**if** $\exists(M', x, y) \in \mathcal{S}'$ s.t. $\mathrm{rowsp}(M - M') \subseteq L'_{\mathsf{all}}$ **return** y	**if** $\exists(M', x, y) \in \mathcal{S}'$ s.t. $\mathrm{rowsp}(M - M') \subseteq L'_{\mathsf{all}}$ **return** y
3:	**if** $\exists(M', x, y) \in \mathcal{S}_{\mathsf{rand}}$ s.t. $\mathrm{rowsp}(M - M') \subseteq L'_{\mathsf{all}}$ **return** y	**if** $(M\mathfrak{g}_{\ell+1}, x) \in \mathcal{R} \cup \mathcal{Q}_{\ell+1}^{\mathsf{User}_1} \cup \mathcal{Q}_{\ell+1}^{\mathsf{Sign}} \cup \mathcal{Q}_{\mathsf{rand}}$ **return** $\mathcal{H}'(M\mathfrak{g}_{\ell+1}, x)$
4:	$y \leftarrow\!\!\$\ \{0,1\}^r$; $\mathcal{S}_{\mathsf{rand}} := \mathcal{S}_{\mathsf{rand}} \cup \{(M, x, y)\}$; **return** y	$\mathcal{H}'(M\mathfrak{g}_{\ell+1}, x) := y \leftarrow\!\!\$\ \{0,1\}^r$; $\mathcal{Q}_{\mathsf{rand}} := \mathcal{Q}_{\mathsf{rand}} \cup \{(M, x)\}$; **return** y

Note that on both sides, due to validity of $\mathcal{S}'$ and L'_{all}, whenever case 2 applies (which is the same on both sides), it always returns the *real* output, i.e., $y = \mathcal{H}(M\mathfrak{g}_{\ell+1}, x)$. The same clearly applies to case 1. Only cases 3 and 4 may return outputs that do not agree with $\mathcal{H}$.
For the sake of contradiction, suppose that there is some query (M, x) on which the two oracles behave differently, and assume that (M, x) is the *first* such query. First suppose that $(M\mathfrak{g}_{\ell+1}, x) \in \mathcal{Q}_{\ell+1}^{\mathsf{Sign}} \cup \mathcal{Q}_{\ell+1}^{\mathsf{User}_1} \cup \mathcal{R}$. In this case we will prove that the *real* output $\mathcal{H}(M\mathfrak{g}_{\ell+1}, x)$ will be returned on both sides (and, by the left side, even when case 1 is not present). There are three cases:

- $(M\mathfrak{g}_{\ell+1}, x) \in \mathcal{Q}_{\ell+1}^{\mathsf{Sign}} \setminus \mathcal{R}$. Unless $\mathsf{BadVerifyQuery}$ holds, the query must have been made inside $\widetilde{\mathsf{User}}_2$ (instead of $\widetilde{\mathsf{Verify}}$). This means that we can assume that step 1 is present on the left-hand side, which therefore must return $y := \mathcal{H}(M\mathfrak{g}_{\ell+1}, x)$. On the right-hand side, either case 2 applies (which always returns the *real* output y), or otherwise case 3 applies, which also returns y by definition.
- $(M\mathfrak{g}_{\ell+1}, x) \in \mathcal{R}$. Note that because (M, x, y) will be added to $\mathcal{S}_{\mathsf{new}}$, unless $\mathsf{LearnedQuery}$ holds, there must be some M' with $M\mathfrak{g}_{\ell+1} = M'\mathfrak{g}_{\ell+1}$ and $(M', x, y) \in \mathcal{S}_{\ell+1}$. Thus, by definition of the vector space $V_{\ell+1}$, we have $\mathrm{rowsp}(M') \subseteq V_{\ell+1}$. As argued before, we always have $\mathrm{rowsp}(M) \subseteq V_{\ell+1} + \langle \mathbf{e}_{n_\ell^{\mathfrak{g}}+1}, \ldots, \mathbf{e}_{n_{\ell+1}^{\mathfrak{g}}} \rangle$ for every query (M, x), which means that

  $$\mathrm{rowsp}(M - M') \subseteq V_{\ell+1} + \langle \mathbf{e}_{n_\ell^{\mathfrak{g}}+1}, \ldots, \mathbf{e}_{n_{\ell+1}^{\mathfrak{g}}} \rangle .$$

  For the sake of contradiction, suppose that case 2 does not find the triple (M', x, y). Then, by definition of $\mathcal{O}_{\mathsf{H}}^{\mathcal{H}'}[\mathfrak{g}_{\ell+1}]$ (with $\mathsf{LearnRel} : (L_{\mathsf{new}}, \mathcal{S}_{\ell+1})$ activated), $\mathrm{rowsp}(M - M')$ would be added to L_{new}. As in the proof for identical behavior of $\mathsf{Sim}\mathcal{O}_{\mathsf{eq}}[L'_{\mathsf{all}}]$ and $\mathcal{O}_{\mathsf{eq}}[\mathfrak{g}_{\ell+1}]$, this implies $\mathrm{rowsp}(M - M') \subseteq L'_{\mathsf{all}}$, i.e., case 2 *does* find the triple (M', x, y), a contradiction.
- $(M\mathfrak{g}_{\ell+1}, x) \in \mathcal{Q}_{\ell+1}^{\mathsf{User}_1}$. During the execution of the $(\ell+1)$-th $\textsc{InitSession}$, a triple (M', x, y) with $M\mathfrak{g}_{\ell+1} = M'\mathfrak{g}_{\ell+1}$ must have been added to $\mathcal{S}_{\ell+1}$. Because all input elements $\textsc{InitSession}$ only have representations in $\langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathsf{vk}}+1} \rangle + \langle \mathbf{e}_{n_\ell^{\mathfrak{g}}+1}, \ldots, \mathbf{e}_{n_{\ell+1}^{\mathfrak{g}}} \rangle \subseteq V_{\ell+1}$, we must have $\mathrm{rowsp}(M') \subseteq V_{\ell+1}$. As in the previous case, this implies that case 2 applies, i.e., $y = \mathcal{H}(M\mathfrak{g}_{\ell+1}, x)$ is returned by all oracles.

Now consider any query (M, x) made by $\widetilde{\mathsf{User}}_2$ or $\widetilde{\mathsf{Verify}}$ for which $(M\mathfrak{g}_{\ell+1}, x) \notin \mathcal{R} \cup \mathcal{Q}_{\ell+1}^{\mathsf{User}_1} \cup \mathcal{Q}_{\ell+1}^{\mathsf{Sign}}$. Here, we may disregard case 1. In addition, since case 2 is identical in both SimForge and Hyb_1, we only need to show that cases 3-4 behave identically for a sequence of RO inputs (M, x) all of which satisfy $(M\mathfrak{g}_{\ell+1}, x) \notin \mathcal{R} \cup \mathcal{Q}_{\ell+1}^{\mathsf{User}_1} \cup \mathcal{Q}_{\ell+1}^{\mathsf{Sign}}$.
Whenever such a query (M, x) is made for which no prior (M', x) with $M\mathfrak{g}_{\ell+1} = M'\mathfrak{g}_{\ell+1}$ has been queried, note that case 3 does not apply. Thus, in both oracles, case 4 returns a uniformly random $y \in \{0,1\}^r$.
If another (M', x) with $M\mathfrak{g}_{\ell+1} = M'\mathfrak{g}_{\ell+1}$ has been queried before, then the oracle in Hyb_1 (using case 3) returns the same output y as before, because $(M' \cdot \mathfrak{g}_{\ell+1}, x)$ was added to $\mathcal{Q}_{\mathsf{rand}}$. Similarly, the oracle in SimForge added the triple (M', x, y) to $\mathcal{S}_{\mathsf{rand}}$. Hence, case 3 finds $(M', x, y) \in \mathcal{S}_{\mathsf{rand}}$ when given the query (M, x) unless $\text{rowsp}(M - M') \not\subseteq L'_{\mathsf{all}}$. However, since $\text{rowsp}(M - M')$ is added to L_{new}, similar to before this would result in a contradiction unless LearnedRelation holds.

To prove the implication InconsistentRO $\Rightarrow$ LearnedRelation, note that InconsistentRO in Hyb_1 means that there have been two queries $(M, x), (M', x)$ with $M\mathfrak{g}_{\ell+1} = M'\mathfrak{g}_{\ell+1}$, s.t. case 2 applied to one of them but not the other. However, this would mean $\text{rowsp}(M - M') \not\subseteq L'_{\mathsf{all}}$, and since $\text{rowsp}(M - M')$ is added to L_{new} this would result in a contradiction unless LearnedRelation holds. □

3.9 Proving success conditioned on Forgeability (Lemma 3.9)

On a high level, note that Forgeability directly implies the existence of a tuple $(L_{\mathsf{forgery}}, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}})$ that results in a successful forgery. However, this does not mean that our adversary computes a valid forgery: our adversary may find a *different* tuple for which the simulated forging probability is high, but then it turns out that in reality it does not result in a forgery. Hence, a crucial observation is that whenever it does not result in a forgery, with large probability we are still going to have some "positive outcome", meaning that we learned some new linear relations or RO input-output pairs. Then, because a sufficient number of forging attempts is performed and there is a limited amount of information to learn, at some point a forgery must be produced.

Consider any fixed execution of $\mathbf{omuf}_{\mathsf{BS},\mathcal{A}}$, which defines a vector $\mathfrak{g}$ of group elements with corresponding length $n^{\mathfrak{g}}$ (these variables refer to the state at the end of the adversary's execution – but note that they never change after running InitSession in the beginning of the forging phase), as well as a message $m_{\ell+1}$ and state $\widetilde{\mathsf{st}}_{\ell+1}$. Let

- $\mathcal{Q}^*$ be an arbitrary set of triples $(\mathfrak{h}, x, y) \in \mathbb{G}^{n^{\mathsf{H}}} \times \{0,1\}^* \times \{0,1\}^r$ where no $(\mathfrak{h}, x)$ appears more than once (we can think of $\mathcal{Q}^*$ as a *partial* RO in which only part of the function table has been fixed so far),
- $L' \subseteq \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}} \rangle$ be an arbitrary valid vector space (w.r.t. $\mathfrak{g}$ and $n^{\mathfrak{g}}$),
- $L_{\mathsf{fake}}, L_{\mathsf{forgery}} \subseteq \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}+n^{\mathsf{msgS}}} \rangle$ be arbitrary vector spaces s.t. $L_{\mathsf{forgery}} \cap \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}} \rangle \subseteq L'$,
- $\mathcal{S}'$ be an arbitrary valid set of RO triples (w.r.t. $\mathcal{Q}^*$, $\mathfrak{g}$, and $n^{\mathfrak{g}}$),
- $\mathcal{S}_{\mathsf{fake}}$ be an arbitrary set of RO triples (which does not have to be valid), and
- s^{msgS} be an arbitrary bitstring.

Further assume that given all of these variables, the following applies:

$$\Pr\left[\text{SimForge}(L' + L_{\mathsf{forgery}}, \mathcal{S}', \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}_{\ell+1}, m_{\ell+1}, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}}) = 1\right] \geq \frac{1}{2}\,. \tag{23}$$

Treating all of this as fixed, we can define the following experiment Exp':

- $\mathcal{H}'$ is a fresh RO, except that $\mathcal{H}'(\mathfrak{h}, x) = y$ for every $(\mathfrak{h}, x, y) \in \mathcal{Q}^*$
- $(L_{\mathsf{new}}, \mathcal{S}_{\mathsf{new}}, \sigma, b) \leftarrow \text{Forge}^{\mathcal{H}'}(\mathfrak{g}, n^{\mathfrak{g}}, L', \mathcal{S}', \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}_{\ell+1}, m_{\ell+1}, L_{\mathsf{forgery}}, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}})$

For Exp', we define two bad events (both of which imply that the adversary learns something):

- Bad_L: In the end, we have $L_{\mathsf{new}} \cap \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}} \rangle \not\subseteq L'$.
- $\mathsf{Bad}_{\mathcal{S}}$: In the end, there is a triple $(M, x, y) \in \mathcal{S}_{\mathsf{new}}$ for which $(M \cdot \mathfrak{g}, x, y) \in \mathcal{Q}^*$ but no $(M', x, y) \in \mathcal{S}'$ with $M \cdot \mathfrak{g} = M' \cdot \mathfrak{g}$.

Now we are ready to state the following lemma, saying that for any tuple that the adversary may find in the forging phase, it likely results in a valid forgery, or in another event that allows us to learn something.

Lemma 3.18. *For any $\mathcal{Q}^*$, $L', L_{\mathsf{forgery}}, L_{\mathsf{fake}}, \mathcal{S}', \mathcal{S}_{\mathsf{fake}}, s^{\mathsf{msgS}}$ fulfilling the criteria described above, we have*

$$\Pr[\mathsf{Exp}' : b = 1 \vee \mathsf{Bad}_L \vee \mathsf{Bad}_{\mathcal{S}}] \geq \frac{1}{2} - \mathrm{negl}(\lambda) \ .$$

Before proving Lemma 3.18, we show how it implies Lemma 3.9.

We now consider a slight modification of our OMUF adversary $\mathcal{A}$ in which *exactly* k_{attempts} iterations are executed in the forging phase, even when a valid forgery has been found earlier. For each $i \in [k_{\mathsf{attempts}}]$, we denote by $\mathcal{Q}_i^*$ the set that contains all RO triples $(\mathfrak{h}, x, y)$ for queries $\mathcal{H}(\mathfrak{h}, x) = y$ that have been made at some point before the i-th forging attempt *by anyone* during the $\mathbf{omuf}_{\mathsf{BS},\mathcal{A}}$-game. In addition, let L_i' be the state of the vector space L at that time (note that this implies $L_i' \supseteq L$ since the space of learned linear relations can only grow), and similarly let $\mathcal{S}_i'$ be the state of $\mathcal{S}$ at that time (for which we have $\mathcal{S}_i' \supseteq \mathcal{S}$). Let the event $\mathsf{FoundTuple}^{(i)}$ be true iff the adversary did find a tuple $(L_{\mathsf{forgery}}, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}})$ in the i-th forging iteration.

We will view the RO $\mathcal{H}$ (sampled inside the game $\mathbf{omuf}_{\mathsf{BS},\mathcal{A}}$) as being *lazy*, i.e., it only samples the output corresponding to some query when it is made for the first time. Thus, the randomness inside the i-th execution of $\textsc{Forge}^{\mathcal{H}}$ (see Figure 7) not only consists of the randomness of $\widetilde{\mathsf{User}}_2$, but also that of fresh RO queries that are not contained in $\mathcal{Q}_i^*$. Thus, whenever $\mathsf{FoundTuple}^{(i)}$ holds, then we may identify the corresponding i-th forging attempt

$$(L_{\mathsf{new}}^{(i)}, \mathcal{S}_{\mathsf{new}}^{(i)}, \sigma^{(i)}, b^{(i)}) \leftarrow \textsc{Forge}^{\mathcal{H}}(\mathfrak{g}, n^{\mathfrak{g}}, L_i', \mathcal{S}_i', \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}_{\ell+1}, m_{\ell+1}, L_{\mathsf{forgery}}, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}}) \ .$$

with an execution of the experiment Exp' (on $\mathcal{Q}_i^*$, L_i', L_{forgery}, L_{fake}, $\mathcal{S}_i'$, $\mathcal{S}_{\mathsf{fake}}$, and s^{msgS}). If we now define (whenever $\mathsf{FoundTuple}^{(i)}$ is true) the event $\mathsf{Bad}_L^{(i)}$ as $L_{\mathsf{new}}^{(i)} \cap \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}} \rangle \not\subseteq L_i'$ and the event $\mathsf{Bad}_{\mathcal{S}}^{(i)}$ as existence of a triple $(M, x, y) \in \mathcal{S}_{\mathsf{new}}^{(i)}$ for which $(M \cdot \mathfrak{g}, x, y) \in \mathcal{Q}^*$ while there is no $(M', x, y) \in \mathcal{S}_i'$ with $M \cdot \mathfrak{g} = M' \cdot \mathfrak{g}$, then *for every fixed state* of $\mathcal{A}$ at the beginning of the i-th forging iteration, we have (for sufficiently large λ):

$$\Pr\left[\ \overline{\mathsf{FoundTuple}^{(i)}} \vee b^{(i)} = 1 \vee \mathsf{Bad}_L^{(i)} \vee \mathsf{Bad}_{\mathcal{S}}^{(i)}\right] \geq \frac{1}{2} - \mathrm{negl}(\lambda) \geq \frac{1}{3} \ ,$$

where the probability is taken over $\textsc{Forge}^{\mathcal{H}}(\mathfrak{g}, n^{\mathfrak{g}}, L_i', \mathcal{S}_i', \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}_{\ell+1}, m_{\ell+1}, L_{\mathsf{forgery}}, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}})$. This follows from Lemma 3.18 for sufficiently large λ.

Let $\mathsf{Good}^{(i)} := \overline{\mathsf{FoundTuple}^{(i)}} \vee b^{(i)} = 1 \vee \mathsf{Bad}_L^{(i)} \vee \mathsf{Bad}_{\mathcal{S}}^{(i)}$. Note that the k_{attempts} events $\mathsf{Good}^{(i)}$ (in the $\mathbf{omuf}_{\mathsf{BS},\mathcal{A}}$ game) are not necessarily independent of each other. Nevertheless, as shown above, for *every* fixed state before the i-th iteration, the event $\mathsf{Good}^{(i)}$ has probability at least $\frac{1}{3}$. Hence, sampling $\mathsf{Good}^{(1)}, \ldots, \mathsf{Good}^{(k_{\mathsf{attempts}})}$ using the game $\mathbf{omuf}_{\mathsf{BS},\mathcal{A}}$ can be no worse than sampling all k_{attempts} variables independently s.t. $\mathsf{Good}^{(i)}$ is 1 with probability $= \frac{1}{3}$. This means that we can still apply a Chernoff bound to obtain the following: Let $N := |\{i \in [k_{\mathsf{attempts}}] \mid \mathsf{Good}^{(i)}\}|$ the number of iterations in which $\mathsf{Good}^{(i)}$ holds. The expected value is $\mathbb{E}[N] \geq k_{\mathsf{attempts}}/3 = 2(n^{\mathfrak{g}} + q_{\mathsf{KeyGen}} + \ell \cdot q_{\mathsf{Sign}}) + 8\lambda$, and therefore

$$\Pr[N \leq n^{\mathfrak{g}} + q_{\mathsf{KeyGen}} + \ell \cdot q_{\mathsf{Sign}}] \leq \Pr[N \leq 0.5 \cdot \mathbb{E}[N]] \leq \frac{1}{e^{\mathbb{E}[N]/8}} \leq \frac{1}{e^{\lambda}} \leq \mathrm{negl}(\lambda) \ ,$$

taken over $\mathbf{omuf}_{\mathsf{BS},\mathcal{A}}$. Note that

- whenever $\mathsf{Bad}_L^{(i)}$ holds for some i, then $\dim(L'_{i+1}) > \dim(L'_i)$ (because $L_{\mathsf{new}}^{(i)} \cap \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}} \rangle$ contains a vector that is not in L'_i but is in $L'_{i+1} = L'_i + (L_{\mathsf{new}}^{(i)} \cap \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}} \rangle)$); and
- whenever $\mathsf{Bad}_{\mathcal{S}}^{(i)}$ holds for some i, then $\mathcal{Q}^*_{i+1} \setminus \mathcal{S}'_{i+1} \subsetneq \mathcal{Q}^*_i \setminus \mathcal{S}'_i$, where we abuse notation to denote by

$$\mathcal{Q} \setminus \mathcal{S} = \{(\mathfrak{h}, x) \in \mathcal{Q} \mid \nexists (M, x, y) \in \mathcal{S} \text{ with } M\mathfrak{g} = \mathfrak{h}\}$$

the set of RO inputs in $\mathcal{Q}$ that do *not* have a matching triple in $\mathcal{S}$.
Note that, intuitively, $\mathcal{Q}^*_i \setminus \mathcal{S}'_i$ denotes the number of RO queries that have been made before the i-th forging attempt by anyone, while $\mathcal{A}$ was not yet aware of this query. Hence, why the strict inclusion $\mathcal{Q}^*_{i+1} \setminus \mathcal{S}'_{i+1} \subsetneq \mathcal{Q}^*_i \setminus \mathcal{S}'_i$ holds can be seen as follows: For every query $(\mathfrak{h}, x) \in \mathcal{Q}^*_{i+1} \setminus \mathcal{Q}^*_i$, there must be some corresponding triple $(M, x, y) \in \mathcal{S}'_{i+1}$, because $\mathcal{A}$ does not call the signing oracle in the forging phase and hence $\mathcal{A}$ itself is the one who initiates any new RO queries which are therefore added to $\mathcal{A}$'s knowledge, i.e., $(\mathfrak{h}, x) \notin \mathcal{Q}^*_{i+1} \setminus \mathcal{S}'_{i+1}$. Furthermore, due to $\mathsf{Bad}_{\mathcal{S}}^{(i)}$, there must be some $(\mathfrak{h}, x) \in \mathcal{Q}^*_i \setminus \mathcal{S}'_i$ for which $\mathcal{A}$ adds a new triple (M, x, y) with $\mathfrak{h} = M\mathfrak{g}$ to $\mathcal{S}_{\mathsf{new}} \subseteq \mathcal{S}'_{i+1}$, i.e., $(\mathfrak{h}, x) \notin \mathcal{Q}^*_{i+1} \setminus \mathcal{S}'_{i+1}$.

Therefore, Bad_L can happen at most $n^{\mathfrak{g}}$ times, and $\mathsf{Bad}_{\mathcal{S}}$ can happen at most $|\mathcal{Q}^*_1| = q_{\mathsf{KeyGen}} + \ell \cdot q_{\mathsf{Sign}}$ times. Thus, whenever $N > n^{\mathfrak{g}} + q_{\mathsf{KeyGen}} + \ell \cdot q_{\mathsf{Sign}}$, there must have been some iteration $i \in [k_{\mathsf{attempts}}]$ for which $b^{(i)} = 1$ or $\mathsf{FoundTuple}^{(i)}$ is false. Furthermore, by definition of $\mathsf{Forgeability}$ (and using the fact that L'_i and $\mathcal{S}'_i$ are valid for each iteration $i \in [k_{\mathsf{attempts}}]$), note that $\mathsf{Forgeability}(\mathfrak{g}, n^{\mathfrak{g}}, L, \mathcal{S}, \mathcal{H}, \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}_{\ell+1}, m_{\ell+1})$ implies $\mathsf{FoundTuple}^{(i)}$ for all i.

Whenever $b^{(i)} = 1$, then the adversary has found a valid signature $\sigma_{\ell+1}$ for $m_{\ell+1}$ in the i-th iteration, i.e., $\mathsf{Verify}(\mathsf{vk}, m_{\ell+1}, \sigma_{\ell+1}) = 1$ (since $b^{(i)} = 1$ means that the execution of Verify inside Forge returned 1 given $\sigma_{\ell+1}$; and Verify is deterministic). Also note that by correctness of the blind signature scheme, we have (taken over $\mathbf{omuf}_{\mathsf{BS},\mathcal{A}}$)

$$\Pr[\forall i \in [\ell] : \mathsf{Verify}(\mathsf{vk}, m_i, \sigma_i) = 1] \geq 1 - \mathrm{negl}(\lambda) \ ,$$

i.e., the signature pairs $(m_1, \sigma_1), \ldots, (m_\ell, \sigma_\ell)$ (which the adversary didn't actively have to forge since the *real* signer messages msgS_i for $i \in [\ell]$ have been received during the learning phase) are all valid.

Since $\mathcal{A}$ wins whenever all pairs $(m_1, \sigma_1), \ldots, (m_{\ell+1}, \sigma_{\ell+1})$ are valid, combining all of the facts above with a union bound, we get

$$\begin{aligned}\Pr[\mathbf{omuf}_{\mathsf{BS},\mathcal{A}} \text{ outputs } 1] &\geq \Pr\left[\begin{array}{l} N > n^{\mathfrak{g}} + q_{\mathsf{KeyGen}} + \ell \cdot q_{\mathsf{Sign}} \wedge \\ \mathsf{Forgeability}(\mathfrak{g}, n^{\mathfrak{g}}, L, \mathcal{S}, \mathcal{H}, \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}_{\ell+1}, m_{\ell+1}) \wedge \\ \mathsf{Verify}(\mathsf{vk}, m_i, \sigma_i) = 1 \ \forall i \in [\ell] \end{array}\right] \\ &\geq \Pr[\mathsf{Forgeability}(\mathfrak{g}, n^{\mathfrak{g}}, L, \mathcal{S}, \mathcal{H}, \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}_{\ell+1}, m_{\ell+1})] - \mathrm{negl}(\lambda)\end{aligned}$$

as desired.

Proof of Lemma 3.18. For notational convenience, whenever we write down a finitely-long vector, it is implicitly padded with an infinite number of 0.

As in Forge, we define $L'_{\mathsf{all}} := L' + L_{\mathsf{forgery}}$ and use A and B to denote matrices with

$$L'_{\mathsf{all}} = \mathrm{rowsp}(-B \mid A) \ .$$

In the following, we use $\mathfrak{g}$ to refer to the state of this group element vector *inside* Forge, i.e., we have

$$\mathfrak{g}[n^{\mathfrak{g}} + 1 \ldots n^{\mathfrak{g}} + n^{\mathsf{msgS}}] = A^+ B \cdot \mathfrak{g}[1 \ldots n^{\mathfrak{g}}] + (A^+ A - I) \cdot \mathbf{z} \cdot \mathfrak{1}$$

for some uniformly random $\mathbf{z} \leftarrow\!\!\$\ \mathbb{Z}_p^{n^{\mathsf{msgS}}}$ sampled within FORGE.

We define another bad event for the experiment Exp' (which we will later prove to have negligible probability, see (27)):

- $\mathsf{Bad}_{\mathbf{z}}$: There is a vector $\mathbf{t}^\mathsf{T} = (\mathbf{r}^\mathsf{T} \mid \mathbf{s}^\mathsf{T}) \in L_{\mathsf{new}}$ (with $\mathbf{r} \in \mathbb{Z}_p^{n^{\mathfrak{g}}}$ and $\mathbf{s} \in \mathbb{Z}_p^{n^{\mathsf{msgS}}}$) (as a result of a call to $\mathcal{O}_{\mathsf{eq}}[\mathfrak{g}]$ or $\mathcal{O}_{\mathsf{H}}^{\mathcal{H}'}[\mathfrak{g}]$), despite $\mathbf{s}^\mathsf{T} \cdot (A^+A - I) \neq \mathbf{0}^\mathsf{T}$.

We first prove the following:

$$\begin{aligned} &\Pr[\mathsf{Exp}' : b = 1 \vee \mathsf{Bad}_L \vee \mathsf{Bad}_{\mathcal{S}} \vee \mathsf{Bad}_{\mathbf{z}}] \\ \geq &\Pr\left[\textsc{SimForge}(L' + L_{\mathsf{forgery}}, \mathcal{S}', \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}_{\ell+1}, m_{\ell+1}, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}}) = 1\right] , \end{aligned} \tag{24}$$

where the first probability is taken over Exp', and the second one of the execution of SIMFORGE. Note that the only differences in the two underlying probabilistic procedures are the oracles that FORGE and SIMFORGE provide to their invocations of $\widetilde{\mathsf{User}}_2$ and $\widetilde{\mathsf{Verify}}$. Hence, it suffices to for us to prove that at least one of the events Bad_L, $\mathsf{Bad}_{\mathcal{S}}$, and $\mathsf{Bad}_{\mathbf{z}}$ holds in Exp, or else that

1. $\mathcal{O}_{\mathsf{eq}}[\mathfrak{g}]$ and $\mathsf{Sim}\mathcal{O}_{\mathsf{eq}}[L'_{\mathsf{all}}]$ behave identically, and that
2. $\mathcal{O}_{\mathsf{H}}^{\mathcal{H}'}[\mathfrak{g}]$ and $\mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L'_{\mathsf{all}}, \mathcal{S}' \cup \mathcal{S}_{\mathsf{rand}}] \prec \mathsf{Random}\mathcal{O}_{\mathsf{H}}$ behave identically. Recall that $\mathcal{H}'$ is uniformly random, except on that it matches the set $\mathcal{Q}^*$. Further note that the latter has $\mathsf{StoreRandRO} : \mathcal{S}_{\mathsf{rand}}$ activated, meaning that any query that is answered by $\mathsf{Random}\mathcal{O}_{\mathsf{H}}$ will, in later RO invocations, be contained in $\mathcal{S}_{\mathsf{rand}}$.

We are now going to prove for each query that is made by $\widetilde{\mathsf{User}}_2$ or $\widetilde{\mathsf{Verify}}$ to the group-equality oracle or to the RO, that the behaviors indeed match (or, alternatively, that one of the bad events becomes true):

1. $\mathcal{O}_{\mathsf{eq}}[\mathfrak{g}]$ and $\mathsf{Sim}\mathcal{O}_{\mathsf{eq}}[L'_{\mathsf{all}}]$.
   Suppose that the group-equality is called on the pair $(\mathbf{t}', \mathbf{t}'')$. We are going to write $\mathbf{t} := \mathbf{t}' - \mathbf{t}''$ as $\mathbf{t}^\mathsf{T} = (\mathbf{r}^\mathsf{T} \mid \mathbf{s}^\mathsf{T})$, with $\mathbf{r} \in \mathbb{Z}_p^{n^{\mathfrak{g}}}$ and $\mathbf{s} \in \mathbb{Z}_p^{n^{\mathsf{msgS}}}$. To show that $\mathcal{O}_{\mathsf{eq}}[\mathfrak{g}]$ and $\mathsf{Sim}\mathcal{O}_{\mathsf{eq}}[L'_{\mathsf{all}}]$ behave identically on $(\mathbf{t}', \mathbf{t}'')$, we need to prove
   $$\mathbf{r}^\mathsf{T} \cdot \mathfrak{g} + \mathbf{s}^\mathsf{T} \cdot (A^+B \cdot \mathfrak{g} + (A^+A - I) \cdot \mathbf{z} \cdot \mathfrak{1}) = \mathfrak{o} \iff (\mathbf{r}^\mathsf{T} \mid \mathbf{s}^\mathsf{T}) \in L'_{\mathsf{all}} . \tag{25}$$
   First suppose $(\mathbf{r}^\mathsf{T} \mid \mathbf{s}^\mathsf{T}) \in L'_{\mathsf{all}}$ (right-hand side). Note that $\mathbf{s}^\mathsf{T} \in \mathrm{rowsp}(A)$, and since we also have $AA^+A = A$ by definition of a weak-left inverse, the equality $\mathbf{s}^\mathsf{T}A^+A = \mathbf{s}^\mathsf{T}$ must hold.
   Hence, we get
   $$\begin{aligned} \mathbf{r}^\mathsf{T} \cdot \mathfrak{g} + \mathbf{s}^\mathsf{T} \cdot (A^+B \cdot \mathfrak{g} + (A^+A - I) \cdot \mathbf{z} \cdot \mathfrak{1}) &= (\mathbf{r}^\mathsf{T} + \mathbf{s}^\mathsf{T} \cdot A^+B) \cdot \mathfrak{g} \\ &= ((\mathbf{r}^\mathsf{T} \mid \mathbf{s}^\mathsf{T}) + (\mathbf{s}^\mathsf{T}A^+B \mid -\mathbf{s}^\mathsf{T})) \cdot \mathfrak{g} . \end{aligned} \tag{26}$$
   We already know (by assumption) that $(\mathbf{r}^\mathsf{T} \mid \mathbf{s}^\mathsf{T}) \in L'_{\mathsf{all}}$. In addition, we have $(\mathbf{s}^\mathsf{T}A^+B \mid -\mathbf{s}^\mathsf{T}) \in L'_{\mathsf{all}}$, because multiplying $\mathbf{s}^\mathsf{T}A^+$ with $\mathrm{rowsp}(B \mid -A)$ (which is contained in L'_{all} by definition) shows that $(\mathbf{s}^\mathsf{T}A^+B \mid -\mathbf{s}^\mathsf{T}A^+A) = (\mathbf{s}^\mathsf{T}A^+B \mid -\mathbf{s}^\mathsf{T})$ is also in L'_{all}. Thus, the term $((\mathbf{r}^\mathsf{T} \mid \mathbf{s}^\mathsf{T}) + (\mathbf{s}^\mathsf{T}A^+B \mid -\mathbf{s}^\mathsf{T}))$ from (26) is in L'_{all}, and (by validity of L'_{all}) multiplying this with $\mathfrak{g}$ yields $\mathfrak{o}$.
   Now suppose that the left-hand side of (25) holds. Then, either $\mathsf{Bad}_{\mathbf{z}}$ applies or else we get $\mathbf{s}^\mathsf{T} \cdot (A^+A - I) = 0$, and therefore $(\mathbf{r}^\mathsf{T} + \mathbf{s}^\mathsf{T}A^+B) \cdot \mathfrak{g} = \mathfrak{o}$. Because of $(\mathbf{r}^\mathsf{T} \mid \mathbf{s}^\mathsf{T})$ being added to L_{new} and (as before) $(\mathbf{s}^\mathsf{T}A^+B \mid -\mathbf{s}^\mathsf{T}) \in L'_{\mathsf{all}}$ we obtain
   $$(\mathbf{r}^\mathsf{T} + \mathbf{s}^\mathsf{T}A^+B \mid \mathbf{0}^\mathsf{T}) \in (L'_{\mathsf{all}} + L_{\mathsf{new}}) .$$
   Recall that $L'_{\mathsf{all}} \cap \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}} \rangle \subseteq L'$. In addition, we either have Bad_L or $L_{\mathsf{new}} \cap \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}} \rangle \subseteq L'$. Combining these three facts with $(\mathbf{r}^\mathsf{T} + \mathbf{s}^\mathsf{T}A^+B \mid \mathbf{0}^\mathsf{T}) \in \langle \mathbf{e}_1, \ldots, \mathbf{e}_{n^{\mathfrak{g}}} \rangle$ yields
   $$(\mathbf{r}^\mathsf{T} + \mathbf{s}^\mathsf{T}A^+B \mid \mathbf{0}^\mathsf{T}) \in L' \subseteq L'_{\mathsf{all}} ,$$
   which combined with $(\mathbf{s}^\mathsf{T}A^+B \mid -\mathbf{s}^\mathsf{T}) \in L'_{\mathsf{all}}$ then proves $(\mathbf{r}^\mathsf{T} \mid \mathbf{s}^\mathsf{T}) \in L'_{\mathsf{all}}$.

2. $\mathcal{O}_{\mathsf{H}}^{\mathcal{H}'}[\mathfrak{g}]$ and $\mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L'_{\mathsf{all}}, \mathcal{S}' \cup \mathcal{S}_{\mathsf{rand}}] \prec \mathsf{Random}\mathcal{O}_{\mathsf{H}}$ (the latter with $\mathsf{StoreRandRO} : \mathcal{S}_{\mathsf{rand}}$ activated). First, suppose a query (M, x) with $(M \cdot \mathfrak{g}, x, y) \in \mathcal{Q}^*$ is made. We know that either $\mathsf{Bad}_{\mathcal{S}}$ holds, or that (by validity of $\mathcal{S}'$) there is some $(M', x, y) \in \mathcal{S}'$ with $M \cdot \mathfrak{g} = M' \cdot \mathfrak{g}$. Analogously to the proof above that shows that the group-equality oracles match, we know that $\mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L'_{\mathsf{all}}, \mathcal{S}' \cup \mathcal{S}_{\mathsf{rand}}]$ finds the triple $(M', x, y) \in \mathcal{S}' \cup \mathcal{S}_{\mathsf{rand}}$ (i.e., for all rows $i \in [n^{\mathsf{H}}]$, we have $M_i - M'_i \in L'_{\mathsf{all}}$) and hence returns y. Hence, both RO implementations return y when given (M, x).
Now suppose that a query (M, x) with $(M \cdot \mathfrak{g}, x, y) \notin \mathcal{Q}^*$ (for all bitstrings y) is made. Again there are two cases:
   - Either no query (M', x) with $M \cdot \mathfrak{g} = M' \cdot \mathfrak{g}$ has been made before. Then, $\mathcal{O}_{\mathsf{H}}^{\mathcal{H}'}$ returns a freshly generated random output, and similarly $\mathsf{Random}\mathcal{O}_{\mathsf{H}}$ also returns a fresh random output (note that $\mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L'_{\mathsf{all}}, \mathcal{S}' \cup \mathcal{S}_{\mathsf{rand}}](M, x)$ must return $\perp$, which again follows analogously to the proof above for group-equality oracles). Hence, both oracles behave identically.
   - Or some (M', x) (with $M \cdot \mathfrak{g} = M' \cdot \mathfrak{g}$) has been queried before and returned y. Again analogously to the proof for group-equality oracles, we know that $\mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L'_{\mathsf{all}}, \mathcal{S}' \cup \mathcal{S}_{\mathsf{rand}}]$ finds the triple $(M', x, y) \in \mathcal{S}' \cup \mathcal{S}_{\mathsf{rand}}$ and hence returns y. Furthermore, $\mathcal{O}_{\mathsf{H}}^{\mathcal{H}'}[\mathfrak{g}]$ must also return the same output as for (M', x) because it implements a *function*.

This concludes the proof of (24).

Next, we prove

$$\Pr[\mathsf{Exp}' : \overline{\mathsf{Bad}_L} \wedge \overline{\mathsf{Bad}_{\mathcal{S}}} \wedge \mathsf{Bad}_{\mathbf{z}}] \leq \mathrm{negl}(\lambda) \; . \tag{27}$$

Note that the proof above shows that the behavior of the oracles supplied by Exp' to $\widetilde{\mathsf{User}}_2$ and $\widetilde{\mathsf{Verify}}$ actually does not depend at all on its choice of $\mathbf{z}$ (i.e., the behavior can be simulated knowing only L'_{all} and $\mathcal{S}'$), unless one of the three events Bad_L, $\mathsf{Bad}_{\mathcal{S}}$, and $\mathsf{Bad}_{\mathbf{z}}$ holds. Hence, it suffices if for any fixed state of Exp' that is about to check $\mathbf{t} \cdot \mathfrak{g} \stackrel{?}{=} 0$ (either in $\mathcal{O}_{\mathsf{eq}}[\mathfrak{g}]$ or $\mathcal{O}_{\mathsf{H}}^{\mathcal{H}'}[\mathfrak{g}]$) and for which none of the bad events hold yet (i.e., so far the behavior was unaffected by the choice of $\mathbf{z}$), we have

$$\Pr_{\mathbf{z}}[\mathbf{r}^{\mathsf{T}} \cdot \mathfrak{g} + \mathbf{s}^{\mathsf{T}} A^+ B \cdot \mathfrak{g} + \mathbf{s}^{\mathsf{T}}(A^+ A - I) \cdot \mathbf{z} \cdot \mathfrak{1} = \mathfrak{o}] \leq \mathrm{negl}(\lambda)$$

whenever $\mathbf{s}^{\mathsf{T}} \cdot (A^+ A - I) \neq \mathbf{0}^{\mathsf{T}}$ (i.e., with only negligible probability, $\mathsf{Bad}_{\mathbf{z}}$ will become true at this point in time of running Exp', assuming that $\mathbf{z}$ is sampled now instead of at the beginning of FORGE). Then, by union bound (over all polynomially-many such invocations inside Exp') we obtain (27). Clearly, the inequality above holds since at least one of the components in $\mathbf{s}^{\mathsf{T}}(A^+ A - I)$ must be non-zero, and therefore the distribution of $\mathbf{s}^{\mathsf{T}}(A^+ A - I) \cdot \mathbf{z}$ is uniformly random in $\mathbb{Z}_p$. Thus, the probability is $= 1/p \leq \mathrm{negl}(\lambda)$.

Combining Equations (23), (24) and (27), we get

$$\begin{aligned}
& \Pr[\mathsf{Exp}' : b = 1 \vee \mathsf{Bad}_L \vee \mathsf{Bad}_{\mathcal{S}}] \\
\geq\; & \Pr[\mathsf{Exp}' : b = 1 \vee \mathsf{Bad}_L \vee \mathsf{Bad}_{\mathcal{S}} \vee \mathsf{Bad}_{\mathbf{z}}] - \Pr[\mathsf{Exp}' : \overline{\mathsf{Bad}_L} \wedge \overline{\mathsf{Bad}_{\mathcal{S}}} \wedge \mathsf{Bad}_{\mathbf{z}}] \\
\geq\; & \Pr[\textsc{SimForge}(L'_{\mathsf{all}}, \mathcal{S}', \widetilde{\mathsf{vk}}, \widetilde{\mathsf{st}}_{\ell+1}, m_{\ell+1}, s^{\mathsf{msgS}}, L_{\mathsf{fake}}, \mathcal{S}_{\mathsf{fake}}) = 1] - \mathrm{negl}(\lambda) \\
\geq\; & \frac{1}{2} - \mathrm{negl}(\lambda) \; ,
\end{aligned}$$

as desired. □

3.10 Extension to ROs with outputs containing group elements

We assumed in Definition 2.2 and hence also in Theorem 3.1 that the random oracle has output space $\{0,1\}^r$. It is possible to extend our result to random oracles with output space $\mathbb{G}^{m^{\mathsf{H}}} \times \{0,1\}^r$, i.e., outputs containing both group elements and bitstrings. In order to avoid adding clutter, we only describe the main changes to the proof of Theorem 3.1 here:

- Whenever adversary $\mathcal{A}$ makes a (real) RO query $\mathcal{H}(\mathfrak{h}, x)$ with corresponding output $(\mathfrak{h}', y)$ (either during *learning* or during *forging*), it obtains m^{H} new group elements whose representation (in terms of $\mathfrak{g}$) it does not know. Hence, $\mathcal{A}$ must append $\mathfrak{h}'$ to $\mathfrak{g}$ (this would happen in $\mathcal{O}^{\mathcal{H}}_{\mathsf{H}}[\mathfrak{g}]$ in Figure 4).
  The set $\mathcal{S}$ of "learned" RO queries does not contain triples (M, x, y) anymore, but instead quadruples (M, x, M', y), where M' denotes the representation of the output group elements. Oracles such as $\mathsf{Sim}\mathcal{O}_{\mathsf{H}}[L, \mathcal{S}]$ (Figure 4) are adjusted accordingly.
- The oracle $\mathsf{Random}\mathcal{O}_{\mathsf{H}}$ that is used by SimForge and simulates random RO outputs (see Figure 5) must now additionally take care of returning the representation matrix $M' \in \mathbb{Z}_p^{m^{\mathsf{H}} \times \infty}$ of uniformly generated group elements $\mathfrak{h}'$. Since we do not need to explicitly construct $\mathfrak{h}'$, we can just sample the first column of T as uniformly random, and set everything else to 0. Then, $\mathsf{Random}\mathcal{O}_{\mathsf{H}}$ returns (T, y) (with $y \leftarrow\!\!\$\ \{0,1\}^r$ as before).
- In Definition 3.7, in Lemma 3.12, and in Lemma 3.15: whenever quantifying over valid vector spaces L' and valid sets of RO queries $\mathcal{S}'$, we would first need to quantify over all possible extensions of $\mathfrak{g}$ that L' and $\mathcal{S}'$ would have to be consistent with. (This is because $\mathfrak{g}$ is allowed to continue growing even after the forging phase has started.)
- The notion of *learning* a relation (Definition 3.13) needs to be adapted: the vector space V_i of "necessary" group elements now also spans the *output* group element representation $\mathrm{rowsp}(M')$ for each quadruple $(M, x, M', y) \in \mathcal{S}_i$ with $(M \cdot \mathfrak{g}, x) \in \mathcal{R}$, in addition to $\mathrm{rowsp}(M)$.
- The proof of *forging* being successful (Lemma 3.9) changes slightly: without the extension, note that $n^{\mathfrak{g}}$ and $\mathfrak{g}$ never change during the forging phase. This meant that $n^{\mathfrak{g}}$ was a fixed polynomial, which allowed us to easily conclude that the event Bad_L (which denotes that $\mathcal{A}$ has found a new linear relation while forging) may happen at most a fixed (polynomial) number of times.
  With the extension, $n^{\mathfrak{g}}$ may increase during the forging phase, and hence it may seem that we may run into the event Bad_L an unlimited number of times. However, the crucial observation here is that given group elements $\mathfrak{h}'$ that are appended to $\mathfrak{g}$ due to an RO query that was *not* previously made by the challenger, w.h.p. $\mathcal{A}$ will never learn any linear relation that non-trivially involves the group elements $\mathfrak{h}'$. (This is because $\mathfrak{h}'$ is chosen uniformly at random, and similar to the hardness of DLog, in the GGM it is highly unlikely to find a non-trivial linear relation involving a uniformly random group element when making only a polynomial number of queries.) Hence, the number of times the event Bad_L occurs w.h.p. does not exceed $n^{\mathfrak{g}} + (q_{\mathsf{KeyGen}} + \ell \cdot q_{\mathsf{Sign}}) \cdot m^{\mathsf{H}}$ (where $n^{\mathfrak{g}}$ denotes the state at the beginning of the forging phase).

Acknowledgments Julia Kastner is supported by the Dutch Research Agenda (NWA) project HAPKIDO (Project No. NWA.1215.18.002), which is financed by the Dutch Research Council (NWO). Tessaro's research was partially supported by NSF grants CNS-2026774, CNS-2154174, CNS-2426905, a gift from Microsoft, and a Stellar Development Foundation Academic Research Award.

References

1. Fighting fraud using partially blind signatures. https://engineering.fb.com/2019/10/16/security/partially-blind-signatures/, accessed: 2024-02-28
2. icloud private relay overview. https://www.apple.com/privacy/docs/iCloud_Private_Relay_Overview_Dec2021.PDF
3. PCM: Click fraud prevention and attribution sent to advertiser. https://webkit.org/blog/11940/pcm-click-fraud-prevention-and-attribution-sent-to-advertiser/, accessed: 2021-09-30
4. Trust tokens. https://developer.chrome.com/docs/privacy-sandbox/trust-tokens/
5. Vpn by google one, explained. https://one.google.com/about/vpn/howitworks
6. Abe, M.: A secure three-move blind signature scheme for polynomially many signatures. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 136–151. Springer, Berlin, Heidelberg (May 2001). https://doi.org/10.1007/3-540-44987-6_9
7. Agrawal, S., Kirshanova, E., Stehlé, D., Yadav, A.: Practical, round-optimal lattice-based blind signatures. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 39–53. ACM Press (Nov 2022). https://doi.org/10.1145/3548606.3560650

8. American National Standards Institute, Inc.: ANSI X9.62 public key cryptography for the financial services industry: the elliptic curve digital signature algorithm (ECDSA) (Nov 16, 2005), https://standards.globalspec.com/std/1955141/ANSI%20X9.62
9. Barak, B., Mahmoody-Ghidary, M.: Lower bounds on signatures from symmetric primitives. In: 48th FOCS. pp. 680–688. IEEE Computer Society Press (Oct 2007). https://doi.org/10.1109/FOCS.2007.38
10. Bellare, M., Namprempre, C., Pointcheval, D., Semanko, M.: The power of RSA inversion oracles and the security of Chaum's RSA-based blind signature scheme. In: Syverson, P.F. (ed.) FC 2001. LNCS, vol. 2339, pp. 319–338. Springer, Berlin, Heidelberg (Feb 2002). https://doi.org/10.1007/3-540-46088-8_25
11. 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). https://doi.org/10.1007/s00145-002-0120-1
12. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) ACM CCS 93. pp. 62–73. ACM Press (Nov 1993). https://doi.org/10.1145/168588.168596
13. Benhamouda, F., Lepoint, T., Loss, J., Orrù, M., Raykova, M.: On the (in)security of ROS. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part I. LNCS, vol. 12696, pp. 33–53. Springer, Cham (Oct 2021). https://doi.org/10.1007/978-3-030-77870-5_2
14. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: High-speed high-security signatures. Journal of Cryptographic Engineering **2**(2), 77–89 (Sep 2012). https://doi.org/10.1007/s13389-012-0027-1
15. Beullens, W., Lyubashevsky, V., Nguyen, N.K., Seiler, G.: Lattice-based blind signatures: Short, efficient, and round-optimal. In: Meng, W., Jensen, C.D., Cremers, C., Kirda, E. (eds.) ACM CCS 2023. pp. 16–29. ACM Press (Nov 2023). https://doi.org/10.1145/3576915.3616613
16. Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In: Desmedt, Y. (ed.) PKC 2003. LNCS, vol. 2567, pp. 31–46. Springer, Berlin, Heidelberg (Jan 2003). https://doi.org/10.1007/3-540-36288-6_3
17. Brandt, N., Hofheinz, D., Klooß, M., Reichle, M.: Tightly-secure blind signatures in pairing-free groups. In: Hanaoka, G., Yang, B.Y. (eds.) ASIACRYPT 2025, Part VI. LNCS, vol. 16250, pp. 337–369. Springer, Singapore (Dec 2025). https://doi.org/10.1007/978-981-95-5119-4_11
18. Catalano, D., Fiore, D., Gennaro, R., Giunta, E.: On the impossibility of algebraic vector commitments in pairing-free groups. In: Kiltz, E., Vaikuntanathan, V. (eds.) TCC 2022, Part II. LNCS, vol. 13748, pp. 274–299. Springer, Cham (Nov 2022). https://doi.org/10.1007/978-3-031-22365-5_10
19. Celi, S., Davidson, A., Valdez, S., Wood, C.A.: Privacy Pass Issuance Protocol. Internet-Draft draft-ietf-privacypass-protocol-16, Internet Engineering Task Force (Oct 2023), https://datatracker.ietf.org/doc/draft-ietf-privacypass-protocol/16/, work in Progress
20. Chairattana-Apirom, R., Tessaro, S., Zhu, C.: Pairing-free blind signatures from CDH assumptions. In: Reyzin, L., Stebila, D. (eds.) CRYPTO 2024, Part I. LNCS, vol. 14920, pp. 174–209. Springer, Cham (Aug 2024). https://doi.org/10.1007/978-3-031-68376-3_6
21. Chaum, D.: Blind signatures for untraceable payments. In: Chaum, D., Rivest, R.L., Sherman, A.T. (eds.) CRYPTO'82. pp. 199–203. Plenum Press, New York, USA (1982). https://doi.org/10.1007/978-1-4757-0602-4_18
22. Chaum, D.: Blind signature system. In: Chaum, D. (ed.) CRYPTO'83. p. 153. Plenum Press, New York, USA (1983). https://doi.org/10.1007/978-1-4684-4730-9_14
23. Chaum, D., Fiat, A., Naor, M.: Untraceable electronic cash. In: Goldwasser, S. (ed.) CRYPTO'88. LNCS, vol. 403, pp. 319–327. Springer, New York (Aug 1990). https://doi.org/10.1007/0-387-34799-2_25
24. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Brickell, E.F. (ed.) CRYPTO'92. LNCS, vol. 740, pp. 89–105. Springer, Berlin, Heidelberg (Aug 1993). https://doi.org/10.1007/3-540-48071-4_7
25. Chen, Y., Lombardi, A., Ma, F., Quach, W.: Does fiat-shamir require a cryptographic hash function? In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part IV. LNCS, vol. 12828, pp. 334–363. Springer, Cham, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84259-8_12
26. Crites, E.C., Komlo, C., Maller, M., Tessaro, S., Zhu, C.: Snowblind: A threshold blind signature in pairing-free groups. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part I. LNCS, vol. 14081, pp. 710–742. Springer, Cham (Aug 2023). https://doi.org/10.1007/978-3-031-38557-5_23
27. Davidson, A., Goldberg, I., Sullivan, N., Tankersley, G., Valsorda, F.: Privacy pass: Bypassing internet challenges anonymously. PoPETs **2018**(3), 164–180 (Jul 2018). https://doi.org/10.1515/popets-2018-0026
28. Denis, F., Jacobs, F., Wood, C.A.: RSA Blind Signatures. RFC 9474 (Oct 2023). https://doi.org/10.17487/RFC9474, https://www.rfc-editor.org/info/rfc9474
29. Döttling, N., Hartmann, D., Hofheinz, D., Kiltz, E., Schäge, S., Ursu, B.: On the impossibility of purely algebraic signatures. In: Nissim, K., Waters, B. (eds.) TCC 2021, Part III. LNCS, vol. 13044, pp. 317–349. Springer, Cham (Nov 2021). https://doi.org/10.1007/978-3-030-90456-2_11

30. Fischlin, M.: Round-optimal composable blind signatures in the common reference string model. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 60–77. Springer, Berlin, Heidelberg (Aug 2006). https://doi.org/10.1007/11818175_4
31. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 33–62. Springer, Cham (Aug 2018). https://doi.org/10.1007/978-3-319-96881-0_2
32. Fuchsbauer, G., Plouviez, A., Seurin, Y.: Blind Schnorr signatures and signed ElGamal encryption in the algebraic group model. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part II. LNCS, vol. 12106, pp. 63–95. Springer, Cham (May 2020). https://doi.org/10.1007/978-3-030-45724-2_3
33. Fujioka, A., Okamoto, T., Ohta, K.: A practical secret voting scheme for large scale elections. In: Seberry, J., Zheng, Y. (eds.) AUSCRYPT'92. LNCS, vol. 718, pp. 244–251. Springer, Berlin, Heidelberg (Dec 1993). https://doi.org/10.1007/3-540-57220-1_66
34. Hanzlik, L., Loss, J., Wagner, B.: Rai-choo! Evolving blind signatures to the next level. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part V. LNCS, vol. 14008, pp. 753–783. Springer, Cham (Apr 2023). https://doi.org/10.1007/978-3-031-30589-4_26
35. Hendrickson, S., Iyengar, J., Pauly, T., Valdez, S., Wood, C.A.: Private Access Tokens. Internet-Draft draft-private-access-tokens-01, Internet Engineering Task Force (Oct 2021), https://datatracker.ietf.org/doc/html/draft-private-access-tokens-01, work in Progress
36. Impagliazzo, R., Rudich, S.: Limits on the provable consequences of one-way permutations. In: 21st ACM STOC. pp. 44–61. ACM Press (May 1989). https://doi.org/10.1145/73007.73012
37. Kastner, J., Loss, J., Xu, J.: On pairing-free blind signature schemes in the algebraic group model. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) PKC 2022, Part II. LNCS, vol. 13178, pp. 468–497. Springer, Cham (Mar 2022). https://doi.org/10.1007/978-3-030-97131-1_16
38. Kastner, J., Nguyen, K., Reichle, M.: Pairing-free blind signatures from standard assumptions in the ROM. In: Reyzin, L., Stebila, D. (eds.) CRYPTO 2024, Part I. LNCS, vol. 14920, pp. 210–245. Springer, Cham (Aug 2024). https://doi.org/10.1007/978-3-031-68376-3_7
39. Kastner, J., Tessaro, S., Zaverucha, G.: Round-optimal pairing-free blind signatures. Cryptology ePrint Archive, Paper 2026/091 (2026), https://eprint.iacr.org/2026/091
40. Katsumata, S., Reichle, M., Sakai, Y.: Practical round-optimal blind signatures in the ROM from standard assumptions. In: Guo, J., Steinfeld, R. (eds.) ASIACRYPT 2023, Part II. LNCS, vol. 14439, pp. 383–417. Springer, Singapore (Dec 2023). https://doi.org/10.1007/978-981-99-8724-5_12
41. Katz, J., Schröder, D., Yerukhimovich, A.: Impossibility of blind signatures from one-way permutations. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 615–629. Springer, Berlin, Heidelberg (Mar 2011). https://doi.org/10.1007/978-3-642-19571-6_37
42. Klooß, M., Reichle, M.: Blind signatures from proofs of inequality. In: Kalai, Y.T., Kamara, S.F. (eds.) CRYPTO 2025, Part VI. LNCS, vol. 16005, pp. 157–189. Springer, Cham (Aug 2025). https://doi.org/10.1007/978-3-032-01887-8_6
43. Klooß, M., Reichle, M., Wagner, B.: Practical blind signatures in pairing-free groups. In: Chung, K.M., Sasaki, Y. (eds.) ASIACRYPT 2024, Part I. LNCS, vol. 15484, pp. 363–395. Springer, Singapore (Dec 2024). https://doi.org/10.1007/978-981-96-0875-1_12
44. Lysyanskaya, A.: Security analysis of RSA-BSSA. In: Boldyreva, A., Kolesnikov, V. (eds.) PKC 2023, Part I. LNCS, vol. 13940, pp. 251–280. Springer, Cham (May 2023). https://doi.org/10.1007/978-3-031-31368-4_10
45. Maurer, U.M.: Abstract models of computation in cryptography (invited paper). In: Smart, N.P. (ed.) 10th IMA International Conference on Cryptography and Coding. LNCS, vol. 3796, pp. 1–12. Springer, Berlin, Heidelberg (Dec 2005). https://doi.org/10.1007/11586821_1
46. Nyberg, K., Rueppel, R.A.: A new signature scheme based on the DSA giving message recovery. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) ACM CCS 93. pp. 58–61. ACM Press (Nov 1993). https://doi.org/10.1145/168588.168595
47. Okamoto, T.: Provably secure and practical identification schemes and corresponding signature schemes. In: Brickell, E.F. (ed.) CRYPTO'92. LNCS, vol. 740, pp. 31–53. Springer, Berlin, Heidelberg (Aug 1993). https://doi.org/10.1007/3-540-48071-4_3
48. Okamoto, T., Ohta, K.: Universal electronic cash. In: Feigenbaum, J. (ed.) CRYPTO'91. LNCS, vol. 576, pp. 324–337. Springer, Berlin, Heidelberg (Aug 1992). https://doi.org/10.1007/3-540-46766-1_27
49. del Pino, R., Katsumata, S.: A new framework for more efficient round-optimal lattice-based (partially) blind signature via trapdoor sampling. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part II. LNCS, vol. 13508, pp. 306–336. Springer, Cham (Aug 2022). https://doi.org/10.1007/978-3-031-15979-4_11
50. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. Journal of Cryptology **13**(3), 361–396 (Jun 2000). https://doi.org/10.1007/s001450010003

51. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 239–252. Springer, New York (Aug 1990). https://doi.org/10.1007/0-387-34805-0_22
52. Schnorr, C.P.: Security of blind discrete log signatures against interactive attacks. In: Qing, S., Okamoto, T., Zhou, J. (eds.) ICICS 01. LNCS, vol. 2229, pp. 1–12. Springer, Berlin, Heidelberg (Nov 2001). https://doi.org/10.1007/3-540-45600-7_1
53. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT'97. LNCS, vol. 1233, pp. 256–266. Springer, Berlin, Heidelberg (May 1997). https://doi.org/10.1007/3-540-69053-0_18
54. Tessaro, S., Zhu, C.: Short pairing-free blind signatures with exponential security. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 782–811. Springer, Cham (May / Jun 2022). https://doi.org/10.1007/978-3-031-07085-3_27
55. Wagner, D.: A generalized birthday problem. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 288–303. Springer, Berlin, Heidelberg (Aug 2002). https://doi.org/10.1007/3-540-45708-9_19
56. Zhandry, M.: To label, or not to label (in generic groups). In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part III. LNCS, vol. 13509, pp. 66–96. Springer, Cham (Aug 2022). https://doi.org/10.1007/978-3-031-15982-4_3