On Tight Security Proofs for Schnorr Signatures

Nils Fleischhacker[1], Tibor Jager*[2], and Dominique Schröder†[3]

[1]Ruhr University Bochum
[2]Paderborn University
[3]Friedrich-Alexander-University Erlangen-Nürnberg

Abstract

The Schnorr signature scheme is the most efficient signature scheme based on the discrete logarithm problem and a long line of research investigates the existence of a *tight* security reduction for this scheme in the random oracle model. Almost all recent works present lower tightness bounds and most recently Seurin (Eurocrypt 2012) showed that under certain assumptions the *non*-tight security proof for Schnorr signatures in the random oracle by Pointcheval and Stern (Eurocrypt 1996) is essentially optimal. All previous works in this direction rule out tight reductions from the (one-more) discrete logarithm problem. In this paper we introduce a new meta-reduction technique, which shows lower bounds for the large and very natural class of *generic* reductions. A generic reduction is independent of a particular representation of group elements. Most reductions in state-of-the-art security proofs have this property. It is desirable, because then the reduction applies generically to any concrete instantiation of the group. Our approach shows *unconditionally* that there is no tight generic reduction from any *natural* non-interactive computational problem Π defined over algebraic groups to breaking Schnorr signatures, unless solving Π is easy.

In an additional application of the new meta-reduction technique, we also *unconditionally* rule out any (even non-tight) generic reduction from *natural* non-interactive computational problems defined over algebraic groups to breaking Schnorr signatures in the non-programmable random oracle model.

*Supported by DFG grant JA 2445/1-1.

†This work was partially supported by the collaborative research project PROMISE (16KIS0763) by the German Federal Ministry of Education and Research (BMBF), by the German research foundation (DFG) through the collaborative research center 1223, and by the state of Bavaria at the Nuremberg Campus of Technology (NCT). NCT is a research cooperation between the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) and the Technische Hochschule Nürnberg Georg Simon Ohm (THN).

1 Motivation

The security of a cryptosystem is nowadays usually confirmed by giving a security proof. Typically, such a proof describes a *reduction* from some (assumed-to-be-)hard computational problem to breaking a defined security property of the cryptosystem. A reduction is considered as *tight*, if the reduction solving the hard computational problem has essentially the same running time and success probability as the attacker on the cryptosystem. Essentially, a tight reduction means that a successful attacker can be turned into an efficient algorithm for the hard computational problem *without* any significant increase in the running time and/or significant loss in the success probability. [1] The tightness of a reduction thus determines the strength of the security guarantees provided by the security proof: A non-tight reduction gives weaker security guarantees than a tight one. Moreover, tightness of the reduction affects the efficiency of the cryptosystem when instantiated in practice: A tighter reduction allows to securely use smaller parameters (shorter moduli, a smaller group size, etc.). Therefore, it is very desirable for a cryptosystem to have a tight security reduction.

In the domain of digital signatures, tight reductions are known for many fundamental schemes, such as Rabin/Williams signatures [5], many strong-RSA-based signatures [28], and RSA Full-Domain Hash [20]. For Schnorr signatures [29, 30], however, the story is a bit different. Schnorr's scheme is one of the most fundamental public-key cryptosystems and Pointcheval and Stern have shown that it is provably secure, assuming the hardness of the discrete logarithm (DL) problem [25] in the Random Oracle Model (ROM) [3]. However, the reduction of Pointcheval and Stern from the discrete logarithm problem to breaking Schnorr signatures is not tight: It loses a factor of q in the time-to-success ratio, where q is the number of random oracle queries performed by the forger.

This has lead to a long line of research investigating the existence of tighter security proofs for Schnorr signatures. At Asiacrypt 2005 Paillier and Vergnaud [24] gave a first lower bound showing that any algebraic reduction (even in the ROM) converting a forger for Schnorr signatures into an algorithm solving the discrete logarithm problem must lose a factor of at least $q^{1/2}$. Their result is quite strong, as they rule out reductions even for adversaries that do not have access to a signing oracle and receive as input the message for which they must forge (UUF-NMA, see Section 3.1 for a formal definition). However, their result also has some limitations: It holds only under the interactive one-more discrete logarithm assumption, they only consider algebraic reductions, and they only rule out tight reductions from the (one-more) discrete logarithm problem. At Crypto 2008 Garg et al. [18] refined this result, by improving the bound from $q^{1/2}$ to $q^{2/3}$ with a new analysis and show that this bound is optimal if the meta-reduction follows a particular approach for simulating the forger. At Eurocrypt 2012 Seurin [31] finally closed the gap between the security proof of Pointcheval and Stern [25] and known impossibility results, by describing a

[1] Usually even a polynomially-bounded increase/loss is considered as significant, if the polynomial may be large. An increase/loss by a small constant factor is not considered as significant.

novel elaborate simulation strategy for the forger and providing a new analysis. All previous works [24, 18, 31] on the existence of tight security proofs for Schnorr signatures have the following in common:

1. They only rule out the existence of tight reductions from specific strong computational problems, namely the (one-more) discrete logarithm problem [2]. Reduction from weaker problems such as, e.g., the computational or decisional Diffie-Hellman problem (CDH/DDH) are not considered.

2. The impossibility results are not unconditional but instead are themselves only valid under the very strong OMDL hardness assumption.

3. They hold only with respect to a limited (but natural) class of reductions, so-called *algebraic reductions*.

It is not entirely unlikely that first the nonexistence of a tight reduction from *strong* computational problems is proven, and later a tight reduction from some *weaker* problem is found. A concrete recent example in the domain of digital signatures where this has happened is RSA Full-Domain Hash (RSA-FDH) [4]. First, at Crypto 2000 Coron [8] described a non-tight reduction from solving the RSA-problem to breaking the security of RSA-FDH, and at Eurocrypt 2002 [9] showed that under certain conditions no tighter reduction from RSA can exist. Later, at Eurocrypt 2012, Kakvi and Kiltz [20] gave a tight reduction from solving a weaker problem, the so-called Phi-Hiding problem. The leverage, used by Kakvi and Kiltz to circumvent the aforementioned impossibility results, was to assume hardness of a weaker computational problem, i.e., making a stronger assumption. As all previous works rule out only tight reductions from strong computational problems such as DL and OMDL, this might happen again with Schnorr signatures and the following question was left open for 25 years:

> *Does a tight security proof for Schnorr signatures exist based on any weaker computational problem?*

2 Contribution

In this work we answer this question in the negative for an overwhelming class of weaker problems, ruling out the existence of tight reductions for virtually all natural non-interactive computational problems defined over abstract algebraic groups. Like previous works, we consider universal unforgeability under no-message attacks (UUF-NMA-security). Moreover, our results hold unconditionally. In contrast to previous works, we consider *generic* reductions instead of algebraic reductions, but we believe that this restriction is marginal: The motivation of considering only algebraic reductions from [24] applies equally to generic reductions. In particular, to the best of our knowledge all known examples of algebraic reductions are also generic.

Our main technical contribution is a new approach for the simulation of a forger in a meta-reduction, i.e., "a reduction against the reduction", which differs from previous works [24, 18, 31] and which allows us to show the following main result:

Theorem 1 (informal)**.** *For almost any* natural *non-interactive computational problem* Π*, there is no tight* generic *reduction from solving* Π *to breaking the universal unforgeability under no-message attacks of Schnorr signatures.*

Technical approach. We begin with the hypothesis that there exists a tight generic reduction $\mathcal{R}$ from some hard non-interactive problem Π to the UUF-NMA-security of Schnorr signatures. Then we show that under this hypothesis there exists an efficient algorithm $\mathcal{M}$, a meta-reduction, which efficiently solves Π. This implies that the hypothesis is false. The meta-reduction $\mathcal{M} = \mathcal{M}^{\mathcal{R}}$ runs $\mathcal{R}$ as a subroutine, by efficiently *simulating* the forger $\mathcal{A}$ for the reduction $\mathcal{R}$.

All previous works in this direction [24, 18, 31] followed essentially the same approach. The difficulty with meta-reductions is that $\mathcal{M} = \mathcal{M}^{\mathcal{R}}$ must efficiently *simulate* the forger $\mathcal{A}$ for $\mathcal{R}$. Previous works resolved this by using a discrete logarithm oracle provided by the OMDL assumption, which allows to efficiently compute valid signatures in the simulation of forger $\mathcal{A}$. This is also the reason why all previous results are only valid under the OMDL assumption, and were only able to rule out reductions from the discrete log or the OMDL problem. To overcome these limitations, a new simulation technique is necessary.

We revisit the simulation strategy of $\mathcal{A}$ applied in known meta-reductions, and put forward a new technique for proving impossibility results. It turns out that considering *generic* reductions provides additional leverage for simulating a successful forger efficiently, essentially by suitably re-programming the group representation while computing valid signatures. The technical challenge is to prove that the reduction remains oblivious to these changes to the group representation during the simulation, except for some negligible probability. We show how to prove this by adopting the "low polynomial degree" proof technique of Shoup [32], which was originally introduced to analyze the complexity of certain algorithms for the discrete logarithm problem, to the setting considered in this paper.

This new approach turns out to be extremely powerful, as it allows to rule out reductions from any non-interactive *representation-invariant* computational problem. Since almost all common hardness assumptions in algebraic groups (e.g., DL, CDH, DDH, DLIN, etc.) are based on representation-invariant computational problems, we are able to rule out tight generic reductions from virtually any natural computational problem, without making any additional assumptions. Even though we apply it specifically to Schnorr signatures, the overall approach is general. We expect that it is applicable to other cryptosystems as well.

Generic reductions vs. algebraic reductions Similar to algebraic reductions, a generic reduction performs only group operations. The main difference is that the sequence of

group operations performed by an algebraic reduction may (but, to our best knowledge, in all known examples does not) depend on the particular representation of group elements. A generic reduction is required to work essentially identical for any representation of group elements. Generic reductions are by definition more restrictive than algebraic ones, we explain below why we do not consider this restriction as very significant.

An obvious question arising with our work is the relation between algebraic and generic reductions. Is a lower bound for generic reductions much less meaningful than a bound for algebraic reductions? We argue that the difference is not very significant. The restriction to algebraic reductions was motivated by the fact most reductions in known security proofs treat the group as a black-box, and thus are algebraic [24, 18, 31]. However, the same motivation applies to generic reductions as well, with exactly the same arguments. In particular, virtually all examples of algebraic reductions in the literature are also generic.

The vast majority of reductions in common security proofs for group-based cryptosystems treats the underlying group as a black-box (i.e., works for any representation of the group), and thus is generic. This is a very desirable feature, because then the cryptosystem can securely be instantiated with *any* group in which the underlying computational problem is hard. In contrast, representation-specific security proofs would require to reprove security for any particular group representation the scheme is used with. Therefore considering generic reductions seems very reasonable.

Generic reductions vs. security proofs in the generic group model. We stress that we model only the reduction $\mathcal{R}$ as a generic algorithm. We do not restrict the forger $\mathcal{A}$ in this way, as commonly done in security proofs in the generic group model. It may not be obvious that this is possible, because $\mathcal{A}$ expects as input group elements in some specific encoding, while $\mathcal{R}$ can only specify them in the form of random encodings. However, the reduction only gets access to the adversary as a blackbox, which means that the adversary is external to the reduction, and the environment in which the reduction runs can easily translate between the encodings used by reduction and adversary. Further, note that while some reduction from a problem Π may be generic, the actual algorithm solving said problem is not $\mathcal{R}$ itself, but the composition of $\mathcal{R}$ and $\mathcal{A}$ which may very well be non-generic. In particular, this means that any results about equivalence of interesting problems in the generic group model do not apply to the reduction. See Section 3.4 and Figure 2 for further explanation.

Generic reductions in the non-programmable random oracle model. An orthogonal question to the one answered in our main result is whether security proofs – even non-tight ones – for Schnorr signatures exist in weaker models. Paillier and Vergnaud analyzed the security of Schnorr Signatures in the standard model [24]. In particular, they presented an impossibility result for security proofs based on algebraic reductions and the discrete logarithm problem. In a similar vein, Fischlin and Fleischhacker [14] presented

a result about the security of Schnorr signatures in the non-programmable random oracle model. Essentially they prove that in the *non-programmable* ROM [15] no reduction from the discrete logarithm problem can exist that potentially invokes the adversary several times but always on the same input. This class is limited, but encompasses all forking-lemma style reductions used to prove Schnorr signatures secure in the programmable ROM.

Both these results suffer from the same shortcomings already discussed earlier. They only show impossibility for the discrete logarithm problem and they are themselves not unconditional, in that they rely on the hardness of the one-more discrete logarithm problem.

By applying our new simulation technique to reductions in the non-programmable random oracle model, we continue this line of research and show the following result

Theorem 2 (informal)**.** *For almost any* natural *non-interactive computational problem* Π*, there is no (even non-tight)* generic *reduction from solving* Π *to breaking the universal unforgeability under no-message attacks of Schnorr signatures in the non-programmable random oracle model.*

Comparison to [16]. The conference version of this work [16] (published at Asiacrypt 2014) claimed that Theorem 1 holds even for *interactive* computational problems. This was incorrect, as pointed out by Kiltz, Masny, and Pan [21], who in fact were able to give a tight security proof based on the hardness of an – arguably somewhat artificial – interactive computational problem. The question of the existence of a tight security reduction based on a *non-interactive* computational problem (which is of course much more desirable) remains open. The present version of this paper corrects the flaw from the conference version, and thus shows the inexistence of such reductions. Moreover, we have extended this version with Theorem 2 that unconditionally rules out any (even non-tight) generic reduction in certain settings.

Further related work. Dodis et al. [10] showed that it is impossible to reduce any computational problem to breaking the security of RSA-FDH in a model where the RSA-group $\mathbb{Z}_N^*$ is modeled as a generic group. This result extends [11]. Coron [9] considered the existence of tight security reductions for RSA-FDH signatures [4]. This result was generalized by Dodis and Reyzin [12] and later refined by Kiltz and Kakvi [20].

In the context of Schnorr signatures, Neven et al. [23] described necessary conditions the hash function must meet in order to provide existential unforgeability under chosen-message attacks (EUF-CMA), and showed that these conditions are sufficient if the forger (not the reduction!) is modeled as a generic group algorithm.

Several works studied the security of the Schnorr signature scheme in the *multi-user* setting, showing essentially that single-user security tightly implies multi-user security [17, 6, 21].

3 Preliminaries

Notation. If S is a set, we write $s \leftarrow\!\!\$\, S$ to denote the action of sampling a uniformly random element s from S. If A is a probabilistic algorithm, we denote with $a \leftarrow\!\!\$\, A$ the action of computing a by running A. We denote with $\emptyset$ the empty string, the empty set, as well as the empty list, the meaning will always be clear from the context. We write $[n]$ to denote the set of integers from 1 to n, i.e., $[n] := \{1, \ldots, n\}$.

3.1 Schnorr Signatures

Let $\mathbb{G}$ be a group of order p with generator g, and let $H : \mathbb{G} \times \{0,1\}^k \to \mathbb{Z}_p$ be a hash function. The Schnorr signature scheme [29, 30] consists of the following efficient algorithms $(\mathsf{KGen}, \mathsf{Sig}, \mathsf{Vf})$.

$\mathsf{KGen}(g)$**:** The key generation algorithm takes as input a generator g of $\mathbb{G}$. It chooses $\mathsf{sk} \leftarrow\!\!\$\, \mathbb{Z}_p$, computes $\mathsf{pk} := g^{\mathsf{sk}}$, and outputs $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Sig}(\mathsf{sk}, m)$**:** The input of the signing algorithm is a private key sk and a message $m \in \{0,1\}^k$. It chooses a random integer $r \leftarrow\!\!\$\, \mathbb{Z}_p$, sets $R := g^r$ as well as $c := \mathsf{H}(R, m)$, and computes $y := \mathsf{sk} \cdot c + r \bmod p$. It outputs $\sigma = (R, y)$.

$\mathsf{Vf}(\mathsf{pk}, m, (R, y))$**:** The verification algorithm outputs the truth value of $g^y \stackrel{?}{=} \mathsf{pk}^c \cdot R$, where $c = \mathsf{H}(R, m)$.

Remark 3. The above variant of Schnorr signatures is obtained by a straightforward application of the Fiat-Shamir heuristic [13] to Schnorr's ID scheme. There exists an equivalent, in practice often more efficient variant of Schnorr signatures, where a signature consists of $\sigma = (c, y)$ (computed exactly as above), and the verification algorithm outputs the truth value of $c \stackrel{?}{=} \mathsf{H}(\mathsf{pk}^c / g^y, m)$. Both variants are equivalent, as one can efficiently (and tightly) convert a valid signature from one variant into a valid signature of the other. In particular, our tightness bounds cover both variants, but the former is slightly more convenient to use in our proofs.

Universal Unforgeability under No-Message Attacks. Consider the following security experiment involving a signature scheme $(\mathsf{KGen}, \mathsf{Sig}, \mathsf{Vf})$, an attacker $\mathcal{A}$, and a challenger $\mathcal{C}$.

1. The challenger $\mathcal{C}$ computes a key-pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\, \mathsf{KGen}(g)$ and chooses a message $m \leftarrow\!\!\$\, \{0,1\}^k$ uniformly at random. It invokes $\mathcal{A}$ on input (pk, m).

2. Eventually, $\mathcal{A}$ stops, outputting a signature σ.

Definition 4. *We say that* $\mathcal{A}$ (ϵ, t)*-breaks the* UUF-NMA*-security of* $(\mathsf{KGen}, \mathsf{Sig}, \mathsf{Vf})$*, if* $\mathcal{A}$ *runs in time at most* t *and*

$$\Pr[\mathcal{A}(\mathsf{pk}, m) = \sigma : \mathsf{Vf}(\mathsf{pk}, m, \sigma) = 1] \geq \epsilon,$$

where randomness is taken over the random choice of pk*,* m*, and* $\mathcal{A}$*'s random coins.*

Note that UUF-NMA-security is a very weak security goal for digital signatures. Since we are going to prove a negative result, this is not a limitation, but instead, makes our result even stronger. In fact, if we rule out reductions from some problem Π to forging signatures in the sense of UUF-NMA, then the impossibility clearly holds for stronger security notions, such as existential unforgeability under adaptive chosen-message attacks [19], too.

3.2 Computational Problems

Let $\mathbb{G}$ be a cyclic group of order p and $g \in \mathbb{G}$ a generator of $\mathbb{G}$. We write $\mathsf{desc}(\mathbb{G}, g)$ to denote the list of group elements $\mathsf{desc}(\mathbb{G}, g) = (g, g^2, \ldots, g^p) \in \mathbb{G}^p$. We say that $\mathsf{desc}(\mathbb{G}, g)$ is the *enumerating description* of $\mathbb{G}$ with respect to g.

Definition 5. *A non-interactive* computational problem Π *in* $\mathbb{G}$ *is specified by two (computationally unbounded) procedures* $\Pi = (\mathcal{G}_\Pi, \mathcal{V}_\Pi)$*, with the following syntax.*

$\mathcal{G}_\Pi(\mathsf{desc}(\mathbb{G}, g))$ *takes as input an enumerating description of* $\mathbb{G}$*, and outputs a state* st *and a problem instance (the challenge)* $C = (C_1, \ldots, C_u, C') \in \mathbb{G}^u \times \{0,1\}^*$*. We assume in the sequel that at least* C_1 *is a generator of* $\mathbb{G}$*.*

$\mathcal{V}_\Pi(\mathsf{desc}(\mathbb{G}, g), st, S, C)$ *takes as input* $(\mathsf{desc}(\mathbb{G}, g), st, C)$ *as defined above, and* $S = (S_1, \ldots, S_w, S') \in \mathbb{G}^w \times \{0,1\}^*$*. It outputs* 0 *or* 1*.*

The exact description and distribution of st, C, S *and the concrete values of* u *and* w *depend on the considered computational problem.*

Definition 6. *An algorithm* $\mathcal{A}$ (ϵ, t)*-solves the non-interactive computational problem* Π *if* $\mathcal{A}$ *has running time at most* t *and wins the following interactive game against a (computationally unbounded) challenger* $\mathcal{C}$ *with probability at least* ϵ*, where the game is defined as follows:*

1. *The challenger* $\mathcal{C}$ *generates an instance of the problem* $(st, C) \leftarrow\!\!\$\, \mathcal{G}_\Pi(\mathsf{desc}(\mathbb{G}, g))$ *and invokes* $\mathcal{A}$ *on input* C*.*
2. *Eventually, algorithm* $\mathcal{A}$ *outputs a candidate solution* S*. The algorithm* $\mathcal{A}$ *wins the game (i.e., solves the computational problem correctly) if and only if* $\mathcal{V}_\Pi(\mathsf{desc}(\mathbb{G}, g), st, C, S) = 1$*.*

Example 7. The *discrete logarithm problem* in $\mathbb{G}$ is specified by the following procedures. $\mathcal{G}_\Pi(\mathsf{desc}(\mathbb{G}, g))$ outputs (st, C) with $st = \emptyset$ and $C = (g, h)$, where $h \leftarrow\!\!\$\, \mathbb{G}$ is a random group element. $\mathcal{V}_\Pi(\mathsf{desc}(\mathbb{G}, g), st, C, S)$ interprets $S = S' \in \{0,1\}^*$ canonically as an integer in $\mathbb{Z}_p$, and outputs 1 iff $h = g^{S'}$.

Example 8. The UUF-NMA-forgery problem for Schnorr signatures in $\mathbb{G}$ with hash function H is specified by the following procedures. $\mathcal{G}_\Pi(\mathsf{desc}(\mathbb{G}, g))$ outputs (st, C) with $st = m$ and $C = (g, \mathsf{pk}, m) \in \mathbb{G}^2 \times \{0,1\}^k$, where $\mathsf{pk} = g^{\mathsf{sk}}$ for $\mathsf{sk} \leftarrow\!\!\$\, \mathbb{Z}_p$ and $m \leftarrow\!\!\$\, \{0,1\}^k$. The verification algorithm $\mathcal{V}_\Pi(\mathsf{desc}(\mathbb{G}, g), st, C, S)$ parses S as $S = (R, y) \in \mathbb{G} \times \mathbb{Z}_p$, sets $c := \mathsf{H}(R, st)$, and outputs 1 if and only if $\mathsf{pk}^c \cdot R = g^y$.

3.3 Representation-Invariant Computational Problems

In our impossibility results given below, we want to rule out the existence of a tight reduction from as large a class of computational problems as possible. Ideally, we want to rule out the existence of a tight reduction from any computational problem that meets Definition 5. However, it is easy to see that this is not achievable in this generality: as Example 8 shows, the problem of forging Schnorr signatures itself is a problem that meets Definition 5. But necessarily there exists a trivial tight reduction from the problem of forging Schnorr signatures to the problem of forging Schnorr signatures! Therefore we need to restrict the class of considered computational problems to exclude such trivial, artificial problems.

We introduce the notion of *representation-invariant* computational problems. This class of problems captures virtually any reasonable computational problem defined over an abstract algebraic group except for a few extremely artificial problems. In particular, the problem of forging Schnorr signatures is *not* contained in this class (see Example 11 below).

Intuitively, a computational problem is *representation-invariant*, if a valid solution to a given problem instance remains valid even if the representation of group elements in challenges and solutions is converted to a different representation of the same group. More formally we define it as follows:

Definition 9. *We say that* Π *is* representation-invariant*, if and only if for all isomorphic groups* $\mathbb{G}, \hat{\mathbb{G}}$ *and for all generators* $g \in \mathbb{G}$*, all* $C = (C_1, \ldots, C_u, C') \leftarrow\!\!\$\, \mathcal{G}_\Pi(\mathsf{desc}(\mathbb{G}, g))$*, all* $st = (st_1, \ldots, st_t, st') \in \mathbb{G}^t \times \{0,1\}^*$*, and all* $S = (S_1, \ldots, S_w, S') \in \mathbb{G}^w \times \{0,1\}^*$ *holds that* $\mathcal{V}_\Pi(\mathsf{desc}(\mathbb{G}, g), st, C, S) = 1 \iff \mathcal{V}_\Pi(\mathsf{desc}(\hat{\mathbb{G}}, \hat{g}), \hat{st}, \hat{C}, \hat{S}) = 1$*, where* $\hat{g} = \phi(g) \in \mathbb{G}'$*,* $\hat{C} = (\phi(C_1), \ldots, \phi(C_u), C')$*,* $\hat{st} = (\phi(st_1), \ldots, \phi(st_t), st')$*, and* $\hat{S} = (\phi(S_1), \ldots, \phi(S_w), S')$*.*

Observe that this definition only demands the *existence* of an isomorphism $\phi : \mathbb{G} \to \hat{\mathbb{G}}$ and not that it is efficiently computable.

Example 10. The discrete logarithm problem is representation-invariant. Let $C = (g, h) \in \mathbb{G}^2$ be a discrete log challenge, with corresponding solution $S' \in \{0,1\}^*$ such that S' canonically interpreted as an integer $S' \in \mathbb{Z}_p$ satisfies $g^{S'} = h \in \mathbb{G}$. Let $\phi : \mathbb{G} \to \hat{\mathbb{G}}$ be an isomorphism, and let $(\hat{g}, \hat{h}) := (\phi(g), \phi(h))$. Then it clearly holds that $\hat{g}^{\hat{S}'} = \hat{h}$, where $\hat{S}' = S'$.

Virtually all common hardness assumptions in algebraic groups are based on representation-invariant computational problems. Popular examples are, for instance, the discrete log problem (DL), computational Diffie-Hellman (CDH), decisional Diffie-Hellman (DDH), decision linear (DLIN), and so on.

Example 11. The UUF-NMA-forgery problem for Schnorr signatures with hash function H is *not* representation-invariant for any hash function H. Let $C = (g, \mathsf{pk}, m) \leftarrow\!\!\$\, \mathcal{G}_\Pi(\mathsf{desc}(\mathbb{G}, g))$ be a challenge with solution $S = (R, y) \in \mathbb{G} \times \mathbb{Z}_p$ satisfying $\mathsf{pk}^c \cdot R = g^y$, where $c := \mathsf{H}(R, m)$.

Let $\hat{\mathbb{G}}$ be a group isomorphic to $\mathbb{G}$, such that $\mathbb{G} \cap \hat{\mathbb{G}} = \emptyset$ (that is, there exists no element of $\hat{\mathbb{G}}$ having the same representation as some element of $\mathbb{G}$).[2] Let $\mathbb{G} \to \hat{\mathbb{G}}$ denote the isomorphism. If there exists any R such that $\mathsf{H}(R, m) \neq \mathsf{H}(\phi(R), m)$ in $\mathbb{Z}_p$ (which holds in particular if H is collision resistant and ϕ efficiently computable), then we have

$$g^y = \mathsf{pk}^{\mathsf{H}(R,m)} \cdot R \qquad \text{but} \qquad \phi(g)^y \neq \phi(\mathsf{pk})^{\mathsf{H}(\phi(R),m)} \cdot \phi(R).$$

Thus, a solution to this problem is valid only with respect to a particular given representation of group elements.

The UUF-NMA-forgery problem of Schnorr signatures is not representation-invariant, because a solution to this problem involves the hash value $\mathsf{H}(R, m)$ that depends on a concrete representation of group element R. We consider such complexity assumptions as rather unnatural, as they are usually very specific to certain constructions of cryptosystems.

Example 12. The hashed Diffie-Hellman (HDH) [1] problem is an example of a non-representation-invariant form of Diffie-Hellman. For a group $\mathbb{G}$ of order p and a hash function $H : \mathbb{G} \to \{0,1\}^\ell$, the HDH problem is defined as follows. $\mathcal{G}_\Pi(\mathsf{desc}(\mathbb{G}, g))$ chooses $u, v \leftarrow\!\!\$\, \mathbb{Z}_p$ and computes $U := g^u$, $V := g^v$. It then chooses a random bit $b \leftarrow\!\!\$\, \{0,1\}$. If $b = 0$ then it outputs the challenge $C := (U, V, H(g^{uv}))$ and the state $st := 0$. If $b = 1$ then it chooses a random value $r \leftarrow\!\!\$\, \{0,1\}^\ell$ and outputs the challenge $C := (U, V, r)$ and the state $st := 1$. The verification algorithm $\mathcal{V}_\Pi(\mathsf{desc}(\mathbb{G}, g), st, C, S)$ outputs 1 if $S = st$ and 0 otherwise.

3.4 Generic Reductions

In this section we recall the notion of *generic groups*, loosely following [32] (cf. also [22, 27], for instance), and define generic (i.e., representation independent) reductions.

Generic groups. Let $(\mathbb{G}, \cdot)$ be a group of order p and $E \subseteq \{0,1\}^{\lceil \log p \rceil}$ be a set of size $|E| = |\mathbb{G}|$. If $g, h \in \mathbb{G}$ are two group elements, then we write $g \div h$ for $g \cdot h^{-1}$. Following [32] we define an *encoding function* as a random injective map $\phi : \mathbb{G} \to E$. We say that an element $e \in E$ is the *encoding* assigned to group element $h \in \mathbb{G}$, if $\phi(h) = e$.

A *generic group algorithm* is an algorithm $\mathcal{R}$ which takes as input $\hat{C} = (\phi(C_1), \ldots, \phi(C_u), C')$, where $\phi(C_i) \in E$ is an encoding of group element C_i for all $i \in [u]$, and $C' \in \{0,1\}^*$ is

[2] Such a group $\hat{\mathbb{G}}$ can trivially be obtained for any group $\mathbb{G}$, for instance by modifying the encoding by prepending a suitable fixed string to each group element, and changing the group law accordingly.

$\mathcal{O}(e, e', \circ)$

$(e, e', \circ) \in E \times E \times \{\cdot, \div\}$
$(i, j) := \textsc{GetIdx}(e, e')$
return $\textsc{Encode}(\mathcal{L}_i^{\mathbb{G}} \circ \mathcal{L}_j^{\mathbb{G}})$

$\textsc{GetIdx}(\vec{e})$

parse $\vec{e}$ **as** $(e_1, \ldots, e_w)$
for $j = 1, \ldots, w$ **do**
  pick first $i \in \left[\left|\mathcal{L}^E\right|\right]$ s.t. $\mathcal{L}_i^E = e_j$
  $i_j := i$
return $(i_1, \ldots, i_w)$

$\textsc{Encode}(G)$

parse G **as** $(G_1, \ldots, G_u)$
for $j = 1, \ldots, u$ **do**
  if $\exists i$ s.t. $\mathcal{L}_i^{\mathbb{G}} = G_j$
    $e_j := \mathcal{L}_i^E$
  else
    $e_j \leftarrow\!\!\$\, E \setminus \mathcal{L}^E$
append e_j **to** $\mathcal{L}^E$
append G_j **to** $\mathcal{L}^{\mathbb{G}}$
return $(e_1, \ldots, e_u)$

Figure 1: Procedures implementing the generic group oracle.

a bit string. The algorithm outputs $\hat{S} = (\phi(S_1), \ldots, \phi(S_w), S')$, where $\phi(S_i) \in E$ is an encoding of group element S_i for all $i \in [w]$, and $S' \in \{0,1\}^*$ is a bit string. In order to perform computations on encoded group elements, algorithm $\mathcal{R}^{=} = \mathcal{R}^{\mathcal{O}}$ may query a *generic group oracle* (or "*group oracle*" for short). This oracle $\mathcal{O}$ takes as input two encodings $e = \phi(G), e' = \phi(G')$ and a symbol $\circ \in \{\cdot, \div\}$, and returns $\phi(G \circ G')$. Note that $(E, \cdot_{\mathcal{O}})$, where $\cdot_{\mathcal{O}}$ denotes the group operation on E induced by oracle $\mathcal{O}$, forms a group which is isomorphic to $(\mathbb{G}, \cdot)$.

It will later be helpful to have a specific implementation of $\mathcal{O}$. We will therefore assume in the sequel that $\mathcal{O}$ internally maintains two lists $\mathcal{L}^{\mathbb{G}} \subseteq \mathbb{G}$ and $\mathcal{L}^E \subseteq E$. These lists define the encoding function ϕ as $\mathcal{L}_i^E = \phi(\mathcal{L}_i^{\mathbb{G}})$, where $\mathcal{L}_i^{\mathbb{G}}$ and $\mathcal{L}_i^E$ denote the i-th element of $\mathcal{L}^{\mathbb{G}}$ and $\mathcal{L}^E$, respectively, for all $i \in \left[\left|\mathcal{L}^{\mathbb{G}}\right|\right]$. Note that from the perspective of a generic group algorithm it makes no difference whether the encoding function is fixed at the beginning or lazily evaluated whenever a new group element occurs. We will assume that the oracle uses lazy evaluation to simplify our discussion and avoid unnecessary steps for achieving polynomial runtime of our meta-reductions. This is implemented by the following procedures.

Procedure Encode takes a list $G = (G_1, \ldots, G_u)$ of group elements as input. It checks for each $G_j \in L$ if an encoding has already been assigned to G_j, i.e., whether there exists an index i such that $\mathcal{L}_i^{\mathbb{G}} = G_j$. If this holds, Encode sets $e_j := \mathcal{L}_i^E$. Otherwise (if no encoding has been assigned to G_j so far), it chooses a fresh and random encoding $e_j \leftarrow\!\!\$\, E \setminus \mathcal{L}^E$. In either case G_j and e_j are appended to $\mathcal{L}^{\mathbb{G}}$ and $\mathcal{L}^E$, respectively, which gradually defines the map ϕ such that $\phi(G_j) = e_j$. Note also that the same group element and encoding may occur multiple times in the list. Finally, the procedure returns the list $(e_1, \ldots, e_u)$ of encodings.

Procedure GetIdx takes a list $(e_1, \ldots, e_w)$ of encodings as input. For each $j \in [w]$ it defines i_j as the smallest[3] index such that $e_j = \mathcal{L}^E_{i_j}$, and returns $(i_1, \ldots, i_w)$.[4]

The lists $\mathcal{L}^{\mathbb{G}}$ and $\mathcal{L}^E$ are initially empty. Then $\mathcal{O}$ calls $(e_1, \ldots, e_u) \leftarrow\!\!\$\ \textsc{Encode}(C_1, \ldots, C_u)$ to determine encodings for all group elements $C_1, \ldots, C_u$ and starts the generic group algorithm on input $\mathcal{R}(e_1, \ldots, e_u, C')$.

The reduction $\mathcal{R}^{\mathcal{O}}$ may now submit queries of the form $(e, e', \circ) \in E \times E \times \{\cdot, \div\}$ to the generic group oracle $\mathcal{O}$. In the sequel we will restrict $\mathcal{R}$ to issue only queries $(e, e', \circ)$ to $\mathcal{O}$ such that $e, e' \in \mathcal{L}^E$. It determines the smallest indices i and j with $e = e_i$ and $e' = e_j$ by calling $(i, j) = \textsc{GetIdx}(e, e')$. Then it computes $\mathcal{L}^{\mathbb{G}}_i \circ \mathcal{L}^{\mathbb{G}}_j$ and returns the encoding $\textsc{Encode}(\mathcal{L}^{\mathbb{G}}_i \circ \mathcal{L}^{\mathbb{G}}_j)$. Furthemore, we require that $\mathcal{R}$ only outputs encodings $\phi(S_i)$ such that $\phi(S_i) \in \mathcal{L}^E$.

Remark 13. We note that the above restrictions are without loss of generality. To explain this, recall that the assignment between group elements and encodings is random. An alternative implementation $\mathcal{O}'$ of $\mathcal{O}$ could, given an encoding $e \notin \mathcal{L}^E$, assign a random group element $G \leftarrow\!\!\$\ \mathbb{G} \setminus \mathcal{L}^{\mathbb{G}}$ to e by appending G to $\mathcal{L}^{\mathbb{G}}$ and e to $\mathcal{L}^E$, in which case $\mathcal{R}$ would obtain an encoding of an independent, new group element. Of course $\mathcal{R}$ can simulate this behavior easily when interacting with $\mathcal{O}$, too.

Generic reductions. Recall that a (fully black-box [26]) reduction from problem Π to problem Σ is an efficient algorithm $\mathcal{R}$ that solves Π, having black-box access to an algorithm $\mathcal{A}$ solving Σ.

In the sequel we consider reductions $\mathcal{R}^{\mathcal{A},\mathcal{O}}$ having black-box access to an algorithm $\mathcal{A}$ as well as to a generic group oracle $\mathcal{O}$. A generic reduction receives as input a challenge $C = (\phi(C_1), \ldots, \phi(C_u), C') \in \mathbb{G}^u \times \{0,1\}^*$ consisting of u encoded group elements and a bit-string C'. $\mathcal{R}$ may perform computations on encoded group elements, by invoking a generic group oracle $\mathcal{O}$ as described above, and interacts with algorithm $\mathcal{A}$ to compute a solution $S = (\phi(S_1), \ldots, \phi(S_w), S') \in \mathbb{G}^w \times \{0,1\}^*$, which again may consist of encoded group elements $\phi(S_1), \ldots, \phi(S_w)$ and a bit-string $S' \in \{0,1\}^*$.

We stress that the adversary $\mathcal{A}$ does not necessarily have to be a generic algorithm. It may not be immediately obvious that a generic reduction can make use of a non-generic adversary, considering that $\mathcal{A}$ might expect a particular encoding of the group elements. However, this is indeed possible. In particular, most reductions in security proofs for cryptosystems that are based on algebraic groups (e.g. [25, 7, 33], to name a few well-known examples) are independent of a particular group representation, and thus generic.

Recall that $\mathcal{R}$ is fully blackbox, i.e., $\mathcal{A}$ is external to $\mathcal{R}$. Thus, the environment in which the reduction runs can easily translate between the two encodings. Consider as an example the reduction shown in Figure 2 that interacts with a non-generic adversary $\mathcal{A}$. We stress

[3] Recall that the same encoding may occur multiple times in $\mathcal{L}^E$.

[4] Note that GetIdxmay receive only encodings $e_1, \ldots, e_w$ which are already contained in $\mathcal{L}^E$, as otherwise

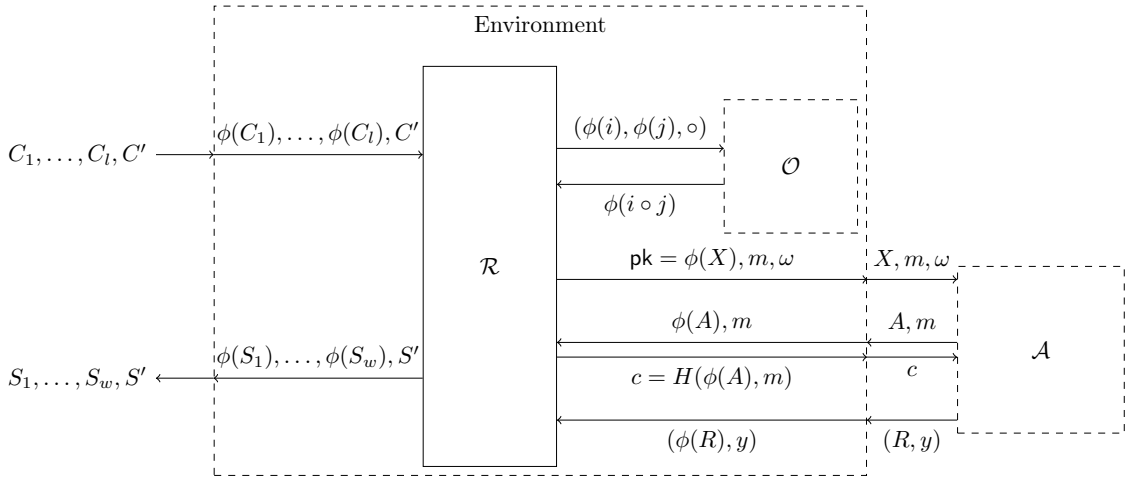

Figure 2: An example of the interaction between a generic reduction $\mathcal{R}$ and a non-generic adversary $\mathcal{A}$ against the unforgeability of Schnorr signatures. All group elements – such as the challenge input, random oracle queries, and the signature output by $\mathcal{A}$ – are encoded by the environment before being passed to $\mathcal{R}$. In the other direction, encodings of group elements output by $\mathcal{R}$ – such as the public key that is the input of $\mathcal{A}$, random oracle responses, and the solution output by $\mathcal{R}$ – are decoded before being passed to the outside world. When a reduction is executed by a metareduction $\mathcal{M}$, the meta-reduction simulates the environment, the group operation oracle and the adversary $\mathcal{A}$. This is indicated by dashed lines.

that the actual algorithm solving the problem Π, which is a composition of $\mathcal{R}$ and $\mathcal{A}$ is therefore *not* generic.

4 Unconditional Tightness Bound for Generic Reductions

In this section, we investigate the possibility of finding a tight *generic* reduction $\mathcal{R}$ that reduces a representation-invariant computational problem Π to breaking the UUF-NMA-security of the Schnorr signature scheme. Our results in this direction are negative, showing that it is impossible to find a generic reduction from any non-interactive representation-invariant computational problem.

4.1 Single-Instance Reductions

We begin with considering a very simple class of reduction that we call *vanilla reductions*. A vanilla reduction is a reduction that runs the UUF-NMA forger $\mathcal{A}$ exactly once (without restarting or rewinding) in order to solve the problem Π. This allows us to explain and analyze the new simulation technique. 342

4.1.1 An Inefficient Adversary $\mathcal{A}$

In this section we describe an inefficient adversary $\mathcal{A}$ that breaks the UUF-NMA-security of the Schnorr signature scheme. Recall that a black-box reduction $\mathcal{R}$ must work for any attacker $\mathcal{A}$. Thus, algorithm $\mathcal{R}^{\mathcal{A}}$ will solve the challenge problem Π, given black-box access to $\mathcal{A}$. The meta-reduction will be able to simulate this attacker *efficiently* for any generic reduction $\mathcal{R}$. We describe this attacker for comprehensibility, in order to make our meta-reduction more accessible to the reader.

1. The input of $\mathcal{A}$ is a Schnorr public-key pk, a message m, and random coins $\omega \in \{0,1\}^{\kappa}$.
2. The forger $\mathcal{A}$ chooses q uniformly random group elements $R_1, \ldots, R_q \leftarrow\!\!\$\, \mathbb{G}$. (We make the assumption that $q \leq |\mathbb{G}|$.) Subsequently, the forger $\mathcal{A}$ queries the random oracle H on (R_i, m) for all $i \in [q]$. Let $c_i := \mathsf{H}(R_i, m) \in \mathbb{Z}_p$ be the corresponding answers.
3. Finally, the forger $\mathcal{A}$ chooses an index uniformly at random $\alpha \leftarrow\!\!\$\, [q]$, computes $y \in \mathbb{Z}_p$ which satisfies the equation $g^y = \mathsf{pk}^{c_\alpha} \cdot R_\alpha$, and outputs (R_α, y). For concreteness, we assume this computation is performed by exhaustive search over all $y \in \mathbb{Z}_p$ (recall that we consider an unbounded attacker here, we show later how to instantiate it efficiently).

Note that (R_α, y) is a valid signature for message m with respect to the public key pk. Thus, the forger $\mathcal{A}$ breaks the UUF-NMA-security of the Schnorr signatures with probability 1.

the behavior of GetIdx is undefined. We will make sure that this is always the case.

4.1.2 Main Result for Vanilla Reductions

Now we are ready to prove our main result for vanilla reductions.

Theorem 14. *Let $\Pi = (\mathcal{G}_\Pi, \mathcal{V}_\Pi)$ be a non-interactive representation-invariant computational problem with a challenge consisting of u group elements and let p be the group order. Suppose there exists a generic vanilla reduction $\mathcal{R}$ that $(\epsilon_\mathcal{R}, t_\mathcal{R})$-solves Π, having* one-time *black-box access to the hypothetical attacker $\mathcal{A}$ described above. Then there exists an algorithm $\mathcal{M}$ that (ϵ, t)-solves Π with $t \approx t_\mathcal{R}$ and*

$$\epsilon \geq \epsilon_\mathcal{R} - \frac{2(u+q+t_\mathcal{R})^2}{p}.$$

Remark 15. The values u, q, and $t_\mathcal{R}$ are polynomially bounded while p is exponential. Therefore, the theorem shows that the existence of a reduction $\mathcal{R}$ implies the existence of a meta-reduction $\mathcal{M}$, which solves Π with essentially the same success probability and running time. Thus, an efficient (and even *non*-tight) reduction $\mathcal{R}$ can only exist if there exists an efficient algorithm for Π, which means that Π cannot be hard.

Proof. Assume that there exists a generic vanilla reduction $\mathcal{R} := \mathcal{R}^{\mathcal{O},\mathcal{A}}$ that $(\epsilon_\mathcal{R}, t_\mathcal{R})$-solves Π, when given access to a generic group oracle $\mathcal{O}$, and a forger $\mathcal{A}(\mathsf{pk}, m, \omega)$, where the inputs to the forger are chosen by $\mathcal{R}$. Furthermore, the reduction $\mathcal{R}$ simulates the random oracle $\mathcal{R}.\mathsf{H}$ for $\mathcal{A}$. We show how to build a meta-reduction $\mathcal{M}$ that has black-box access to $\mathcal{R}$ and solves the representation-invariant problem Π directly.

We describe $\mathcal{M}$ in a sequence of games, beginning with an *inefficient* implementation $\mathcal{M}_0$ of $\mathcal{M}$ and modify it gradually until we obtain an *efficient* implementation $\mathcal{M}_2$ of $\mathcal{M}$. We bound the probability with which any reduction $\mathcal{R}$ can distinguish each implementation $\mathcal{M}_i$ from $\mathcal{M}_{i-1}$ for all $i \in \{1, 2\}$, which yields that $\mathcal{M}_2$ is an efficient algorithm that can use $\mathcal{R}$ to solve Π if $\mathcal{R}$ is tight. In what follows let X_i denote the event that $\mathcal{R}$ outputs a valid solution to the given problem instance $\hat{C}$ of Π in Game i.

Game 0. Our meta-reduction $\mathcal{M}_0$ is an algorithm for solving a representation-invariant computational problem Π, as defined in Section 3.3. That is, $\mathcal{M}_0$ takes as input an instance $C = (C_1, \ldots, C_u, C') \in \mathbb{G}^u \times \{0,1\}^*$, of the representation-invariant computational problem Π and outputs a candidate solution S. $\mathcal{R}$ is a generic reduction, i.e., a representation-independent algorithm for Π having black-box access to an attacker $\mathcal{A}$. Algorithm $\mathcal{M}_0$ runs reduction $\mathcal{R}$ as a subroutine, by simulating the generic group oracle $\mathcal{O}$ and attacker $\mathcal{A}$ for $\mathcal{R}$. In order to provide the generic group oracle for $\mathcal{R}$, $\mathcal{M}_0$ implements the following procedures (cf. Figure 3).

Initialization of $\mathcal{M}_0$: At the beginning of the game, $\mathcal{M}_0$ initializes two lists $\mathcal{L}^\mathbb{G} := \emptyset$ and $\mathcal{L}^E := \emptyset$, which are used to simulate the generic group oracle $\mathcal{O}$. Furthermore, $\mathcal{M}_0$

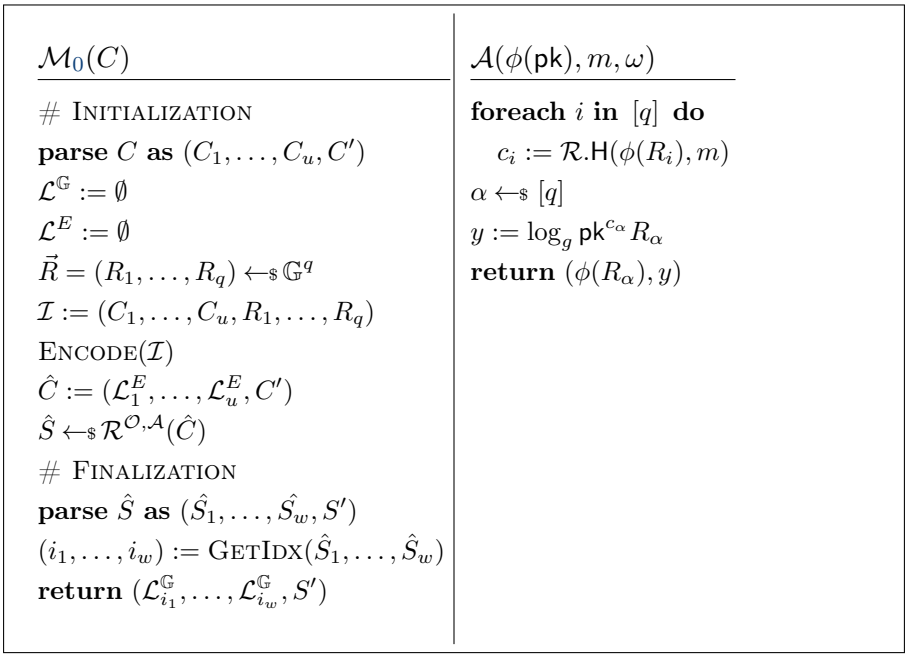

Figure 3: Implementation of $\mathcal{M}_0$.

chooses $\vec{R} = (R_1, \ldots, R_q) \leftarrow\!\!\$\ \mathbb{G}^q$ at random (these values will later be used by the simulated attacker $\mathcal{A}$), sets $\mathcal{I} := (C_1, \ldots, C_u, R_1, \ldots, R_q)$, and runs Encode($\mathcal{I}$) to assign encodings to these group elements. Then $\mathcal{M}_0$ invokes the reduction $\mathcal{R}$ on input $\hat{C} := (\mathcal{L}^E_1, \ldots, \mathcal{L}^E_u, C')$. Note that $\hat{C}$ is an encoded version of the challenge instance of Π received by $\mathcal{M}_0$. That is, we have $\hat{C} = (\phi(C_1), \ldots, \phi(C_u), C')$. Oracle queries of $\mathcal{R}$ are answered by $\mathcal{M}_0$ as follows.

Generic group oracle $\mathcal{O}(e, e', \circ)$: To simulate the generic group oracle, $\mathcal{M}_0$ implements procedures Encode and GetIdx as described in Section 3.4. Whenever $\mathcal{R}$ submits a query $(e, e', \circ) \in E \times E \times \{\cdot, \div\}$ to the generic group oracle $\mathcal{O}$, the meta-reduction determines the smallest indices i and j such that $e = \mathcal{L}^{\mathbb{G}}_i$ and $e' = \mathcal{L}^{\mathbb{G}}_j$ by calling $(i, j) =$ GetIdx(e, e'). Then it computes $\mathcal{L}^{\mathbb{G}}_i \circ \mathcal{L}^{\mathbb{G}}_j$ and returns Encode$(\mathcal{L}^{\mathbb{G}}_i \circ \mathcal{L}^{\mathbb{G}}_j)$.

The forger $\mathcal{A}(\phi(\mathsf{pk}), m, \omega)$: This procedure implements a simulation of the inefficient attacker $\mathcal{A}$ described in Section 4.1.1. It proceeds as follows. When $\mathcal{R}$ outputs $(\phi(\mathsf{pk}), m, \omega)$ to invoke an instance of $\mathcal{A}$, $\mathcal{A}$ queries the random oracle $\mathcal{R}.\mathsf{H}$ provided by $\mathcal{R}$ to determine $c_i = \mathsf{H}(\phi(R_i), m)$ for all $i \in [q]$. Afterwards, $\mathcal{M}_0$ chooses an index $\alpha \leftarrow\!\!\$\ [q]$ uniformly at random, computes the discrete logarithm $y := \log_g \mathsf{pk}^{c_\alpha} R_\alpha$ by exhaustive search, and outputs $(\phi(R_\alpha), y)$. (This step is not efficient. We show in subsequent games how to implement this simulation efficiently.)

$\mathcal{M}_1(C)$

Initialization
parse C **as** $(C_1, \ldots, C_u, C')$
$\mathcal{L}^{\mathbb{G}} := \emptyset$
$\mathcal{L}^{E} := \emptyset$
$\mathcal{L}^{V} := \emptyset$
$\vec{R} = (R_1, \ldots, R_q) \leftarrow_{\$} \mathbb{G}^q$
$\mathcal{I} := (C_1, \ldots, C_u, R_1, \ldots, R_q)$
Encode($\mathcal{I}$)
$\mathcal{L}_i^V := \eta_i,\ \forall i \in [u+q]\,.$
$\hat{C} := (\mathcal{L}_1^E, \ldots, \mathcal{L}_u^E, C')$
$\hat{S} \leftarrow_{\$} \mathcal{R}^{\mathcal{O},\mathcal{A}}(\hat{C})$
Finalization
parse $\hat{S}$ **as** $(\hat{S}_1, \ldots, \hat{S}_w, S')$
$(i_1, \ldots, i_w) :=$ GetIdx$(\hat{S}_1, \ldots, \hat{S}_w)$
return $(\mathcal{L}_{i_1}^{\mathbb{G}}, \ldots, \mathcal{L}_{i_w}^{\mathbb{G}}, S')$

$\mathcal{O}(e, e', \circ)$

$(e, e', \circ) \in E \times E \times \{\cdot, \div\}$
$(i, j) :=$ GetIdx(e, e')
$a := \mathcal{L}_i^V \diamond \mathcal{L}_j^V \in \mathbb{Z}_p^{u+q}$
append a **to** $\mathcal{L}^V$
return Encode$(\mathcal{L}_i^{\mathbb{G}} \circ \mathcal{L}_j^{\mathbb{G}})$

Figure 4: Meta-Reduction $\mathcal{M}_1$. Elements highlighted in gray show the differences to $\mathcal{M}_0$. All other procedures are identical to $\mathcal{M}_0$ and thus omitted.

Finalization of $\mathcal{M}_0$: Eventually, the algorithm $\mathcal{R}$ outputs a solution $\hat{S} := (\hat{S}_1, \ldots, \hat{S}_w, S') \in E^w \times \{0,1\}^*$. The algorithm $\mathcal{M}_0$ runs $(i_1, \ldots, i_w) :=$ GetIdx$(\hat{S}_1, \ldots, \hat{S}_w)$ to determine the indices of group elements $(\mathcal{L}_{i_1}^{\mathbb{G}}, \ldots, \mathcal{L}_{i_w}^{\mathbb{G}})$ corresponding to encodings $(\hat{S}_1, \ldots, \hat{S}_w)$, and outputs $(\mathcal{L}_{i_1}^{\mathbb{G}}, \ldots, \mathcal{L}_{i_w}^{\mathbb{G}}, S')$.

Analysis of $\mathcal{M}_0$. Note that $\mathcal{M}_0$ provides a perfect simulation of the oracle $\mathcal{O}$ and it also mimics the attacker from Section 4.1.1 perfectly. In particular, (R_α, y) is a valid forgery for message m and thus, $\mathcal{R}$ outputs a solution $\hat{S} = (\hat{S}_1, \ldots, \hat{S}_w, S')$ to $\hat{C}$ with probability $\Pr[X_0] = \epsilon_{\mathcal{R}}$. Since Π is assumed to be representation-invariant, $S := (S_1, \ldots, S_w, S')$ with $\hat{S}_i = \phi(S_i)$ for $i \in [w]$ is therefore a valid solution to C. Thus, $\mathcal{M}_0$ outputs a valid solution S to C with probability $\epsilon_{\mathcal{R}}$.

Game 1. In this game we introduce a meta-reduction $\mathcal{M}_1$, which essentially extends $\mathcal{M}_0$ with additional bookkeeping to record the sequence of group operations performed by $\mathcal{R}$. The purpose of this intermediate game is to simplify our analysis of the final implementation $\mathcal{M}_2$. Meta-reduction $\mathcal{M}_1$ proceeds identical to $\mathcal{M}_0$, except for a few differences (cf. Figure 4).

Initialization of $\mathcal{M}_1$: The initialization is exactly as before, except that $\mathcal{M}_1$ maintains an additional list $\mathcal{L}^V$ of elements of $\mathbb{Z}_p^{u+q}$. Let $\mathcal{L}_i^V$ denote the i-th entry of $\mathcal{L}^V$.

List $\mathcal{L}^V$ is initialized with the $u+q$ canonical unit vectors in $\mathbb{Z}_p^{u+q}$. That is, let η_i denote the i-th canonical unit vector in $\mathbb{Z}_p^{u+q}$, i.e.,

$$\eta_1 := (1, 0, \ldots, 0), \eta_2 := (0, 1, 0, \ldots, 0), \ldots, \eta_{u+q} := (0, \ldots, 0, 1).$$

Then $\mathcal{L}^V$is initialized such that $\mathcal{L}_i^V := \eta_i$ for all $i \in [u+q]$.

Generic group oracle $\mathcal{O}(e, e', \circ)$ In parallel to computing the group operation, the generic group oracle implemented by $\mathcal{M}_1$ also performs computations on vectors of $\mathcal{L}^V$.

Given a query $(e, e', \circ) \in E \times E \times \{\cdot, \div\}$, the oracle $\mathcal{O}$ determines the smallest indices i and j such that $e = \mathcal{L}_i^{\mathbb{G}}$ and $e' = \mathcal{L}_j^{\mathbb{G}}$ by calling GetIdx. It computes $a := \mathcal{L}_i^V \diamond \mathcal{L}_j^V \in \mathbb{Z}_p^{u+q}$, where $\diamond := +$ if $\circ = \cdot$ and $\diamond := -$ if $\circ = \div$, and appends a to $\mathcal{L}^V$. Finally it returns $\textsc{Encode}(\mathcal{L}_i^{\mathbb{G}} \circ \mathcal{L}_j^{\mathbb{G}})$.

Analysis of $\mathcal{M}_1$. Recall that the initial content $\mathcal{I}$ of $\mathcal{L}^{\mathbb{G}}$ is $\mathcal{I} = (C_1, \ldots, C_u, R_1, \ldots, R_q)$, and that $\mathcal{R}$ performs only group operations on $\mathcal{I}$. Thus, any group element $h \in \mathcal{L}^{\mathbb{G}}$ can be written as $h = \prod_{i=1}^{u} C_i^{a_i} \cdot \prod_{i=1}^{q} R_i^{a_{u+i}}$ where the vector $a = (a_1, \ldots, a_{u+q}) \in \mathbb{Z}_p^{u+q}$ is (essentially) determined by the sequence of queries issued by $\mathcal{R}$ to $\mathcal{O}$. For a vector $a \in \mathbb{Z}_p^{u+q}$ and a vector of group elements $V = (v_1, \ldots, v_{u+q}) \in \mathbb{G}^{u+q}$ let us write $\mathsf{Eval}(V, a)$ as a shorthand for $\mathsf{Eval}(V, a) := \prod_{i=1}^{u+q} v_i^{a_i}$ in the following. In particular, it holds that $\mathsf{Eval}(\mathcal{I}, a) = \prod_{i=1}^{u} C_i^{a_i} \cdot \prod_{i=1}^{q} R_i^{a_{u+i}}$. The key motivation for the changes introduced in Game 1 is that now (by construction of $\mathcal{M}_1$) it holds that $\mathcal{L}_i^{\mathbb{G}} = \mathsf{Eval}(\mathcal{I}, \mathcal{L}_i^V)$ for all $i \in \left[\left|\mathcal{L}^{\mathbb{G}}\right|\right]$. Thus, at any point in time during the execution of $\mathcal{R}$, the entire list $\mathcal{L}^{\mathbb{G}}$ of group elements can be recomputed from $\mathcal{L}^V$ and $\mathcal{I}$ by setting $\mathcal{L}_i^{\mathbb{G}} := \mathsf{Eval}(\mathcal{I}, \mathcal{L}_i^V)$ for $i \in \left[\left|\mathcal{L}^V\right|\right]$. The reduction $\mathcal{R}$ is completely oblivious to this additional bookkeeping performed by $\mathcal{M}_1$, thus we have $\Pr[X_1] = \Pr[X_0]$.

Game 2. Note that the meta-reductions described in previous games were not efficient, because the simulation of the attacker in procedure $\mathcal{A}$ needed to compute a discrete logarithm by exhaustive search. In this final game, we construct a meta-reduction $\mathcal{M}_2$ that simulates $\mathcal{A}$ efficiently. $\mathcal{M}_2$ proceeds exactly like $\mathcal{M}_1$, except for the following (cf. Figure 5).

The forger $\mathcal{A}(\phi(\mathsf{pk}), m, \omega)$: When $\mathcal{R}$ outputs $(\phi(\mathsf{pk}), m, \omega)$ to invoke an instance of $\mathcal{A}$, $\mathcal{A}$ queries the random oracle $\mathcal{R}.\mathsf{H}$ provided by $\mathcal{R}$ to determine $c_i = \mathsf{H}(\phi(R_i), m)$ for all $i \in [q]$. Afterwards, it proceeds as follows:

- $\mathcal{A}$ chooses an index $\alpha \leftarrow\!\!\$\ [q]$ uniformly at random, samples an element y uniformly at random from $\mathbb{Z}_p$.

$\mathcal{A}(\phi(\mathsf{pk}), m, \omega)$

foreach i **in** $[q]$ **do**
  $c_i = \mathcal{R}.\mathsf{H}(\phi(R_i), m)$
$\alpha \leftarrow\!\!\$\ [q]$
$y \leftarrow\!\!\$\ \mathbb{Z}_p \quad ; \quad R^*_\alpha := g^y \mathsf{pk}^{-c_\alpha}$
$\mathcal{I}^* := (C_1, \ldots, C_u, R_1, \ldots, R_{\alpha-1}, R^*_\alpha, R_{\alpha+1}, \ldots, R_q)$
for $j = 1, \ldots, \left|\mathcal{L}^{\mathbb{G}}\right|$ **do**
  $\mathcal{L}^{\mathbb{G}}_i := \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_i)$
return $(\phi(R^*_\alpha), y)$

Figure 5: Efficient simulation of attacker $\mathcal{A}$ by $\mathcal{M}_2$.

- Then it computes $R^*_\alpha := g^y \mathsf{pk}^{-c_\alpha}$, and re-computes the entire list $\mathcal{L}^{\mathbb{G}}$ using R^*_α instead of R_α.

  More precisely, let $\mathcal{I}^* := (C_1, \ldots, C_u, R_1, \ldots, R_{\alpha-1}, R^*_\alpha, R_{\alpha+1}, \ldots, R_q)$. Observe that the vector $\mathcal{I}^*$ is identical to the initial contents $\mathcal{I}$ of $\mathcal{L}^{\mathbb{G}}$, with the difference that R_α is replaced by R^*_α. The list $\mathcal{L}^{\mathbb{G}}$ is now recomputed from $\mathcal{L}^V$ and $\mathcal{I}^*$ by setting $\mathcal{L}^{\mathbb{G}}_i := \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_i)$ for all $i \in \left[\left|\mathcal{L}^V\right|\right]$.

- Finally, $\mathcal{M}_2$ returns $(\phi(R^*_\alpha), y)$ to $\mathcal{R}$ as the forgery.

Analysis of $\mathcal{M}_2$. First note that $(\phi(R^*_\alpha), y)$ is a valid signature, since $\phi(R^*_\alpha)$ is the encoding of group element R^*_α satisfying the verification equation $g^y = \mathsf{pk}^{c_\alpha} \cdot R^*_\alpha$, where $c_\alpha = \mathsf{H}(\phi(R^*_\alpha), m)$. Next we claim that $\mathcal{R}$ is not able to distinguish $\mathcal{M}_2$ from $\mathcal{M}_1$, except for a negligibly small probability. To show this, observe that Game 2 and Game 1 are perfectly indistinguishable, if for all pairs of vectors $\mathcal{L}^V_i, \mathcal{L}^V_j \in \mathcal{L}^V$ it holds that $\mathsf{Eval}(\mathcal{I}, \mathcal{L}^V_i) = \mathsf{Eval}(\mathcal{I}, \mathcal{L}^V_j) \iff \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_i) = \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_j)$, because in this case $\mathcal{M}_2$ chooses identical encodings for two group elements $\mathcal{L}^{\mathbb{G}}_i, \mathcal{L}^{\mathbb{G}}_j \in \mathcal{L}^{\mathbb{G}}$ *if and only if* $\mathcal{M}_1$ chooses identical encodings. It remains to show that this happens with overwhelming probability. We state this in the following Lemma.

Lemma 16. *Let F denote the event that $\mathcal{R}$ computes vectors $\mathcal{L}^V_i, \mathcal{L}^V_j \in \mathcal{L}^V$ such that*

$$\mathsf{Eval}(\mathcal{I}, \mathcal{L}^V_i) = \mathsf{Eval}(\mathcal{I}, \mathcal{L}^V_j) \quad \wedge \quad \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_i) \neq \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_j) \tag{1}$$

or

$$\mathsf{Eval}(\mathcal{I}, \mathcal{L}^V_i) \neq \mathsf{Eval}(\mathcal{I}, \mathcal{L}^V_j) \quad \wedge \quad \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_i) = \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_j). \tag{2}$$

Then

$$\Pr[F] \leq 2(u + q + t_{\mathcal{R}})^2/p.$$

The proof of Lemma 16 is deferred to Section 4.2. We apply it to finish the proof of Theorem 14. By Lemma 16, algorithm $\mathcal{M}_2$ fails to simulate $\mathcal{M}_1$ with probability at most $2(u+q+t_{\mathcal{R}})^2/p$. Thus, we have $\Pr[X_2] \geq \Pr[X_1] - 2(u+q+t_{\mathcal{R}})^2/p$.

Note also that $\mathcal{M}_2$ provides an efficient simulation of adversary $\mathcal{A}$. The total running time of $\mathcal{M}_2$ is essentially of the running time of $\mathcal{R}$ plus some minor additional computations and bookkeeping. Furthermore, if $\mathcal{R}$ is able to $(\epsilon_{\mathcal{R}}, t_{\mathcal{R}})$-solve Π, then $\mathcal{M}_2$ is able to (ϵ, t)-solve Π with probability at least

$$\epsilon \geq \Pr[X_2] \geq \epsilon_{\mathcal{R}} - \frac{2(u+q+t_{\mathcal{R}})^2}{p}.$$

□

4.2 Proof of Lemma 16

The proof of this lemma is based on the observation that an algorithm that performs only a (polynomially) limited number of group operations in an (exponential-size) generic group is very unlikely to find any "non-trivial relation" among random group elements. This technique was introduced in [32] in a different setting, to analyze the complexity of algorithms for the discrete logarithm problem.

Proof. We first introduce an alternative formulation of event F. Recall that the vectors $\mathcal{I}$ and $\mathcal{I}^*$ differ only in their α-th component. In the sequel let us write $\mathcal{I}_\alpha$ to denote the vector $\mathcal{I}$, but with its α-th component R_α set equal to $1 \in \mathbb{G}$. That is,

$$\mathcal{I}_\alpha := (R_1, \ldots, R_{\alpha-1}, 1, R_{\alpha+1}, \ldots, R_q, g_1, \ldots, g_u).$$

Then we have

$$\mathsf{Eval}(\mathcal{I}, \mathcal{L}_i^V) = \mathsf{Eval}(\mathcal{I}_\alpha, \mathcal{L}_i^V) \cdot R_\alpha^{\mathcal{L}_{i,\alpha}^V} \quad \text{and} \quad \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}_i^V) = \mathsf{Eval}(\mathcal{I}_\alpha, \mathcal{L}_i^V) \cdot (R_\alpha^*)^{\mathcal{L}_{i,\alpha}^V}$$

where $\mathcal{L}_{i,\alpha}^V$ denotes the α-th component of vector $\mathcal{L}_i^V$. In particular, for any two vectors $\mathcal{L}_i^V, \mathcal{L}_j^V$ we have

$$\begin{aligned} \mathsf{Eval}(\mathcal{I}, \mathcal{L}_i^V) = \mathsf{Eval}(\mathcal{I}, \mathcal{L}_j^V) &\iff \mathsf{Eval}(\mathcal{I}_\alpha, \mathcal{L}_i^V) \cdot R_\alpha^{\mathcal{L}_{i,\alpha}^V} = \mathsf{Eval}(\mathcal{I}_\alpha, \mathcal{L}_j^V) \cdot R_\alpha^{\mathcal{L}_{j,\alpha}^V} \\ &\iff \mathsf{Eval}(\mathcal{I}_\alpha, \mathcal{L}_i^V - \mathcal{L}_j^V) \cdot R_\alpha^{\mathcal{L}_{i,\alpha}^V - \mathcal{L}_{j,\alpha}^V} = 1 \end{aligned}$$

Thus, Equation 1 is equivalent to

$$\mathsf{Eval}(\mathcal{I}_\alpha, \mathcal{L}_i^V - \mathcal{L}_j^V) \cdot R_\alpha^{\mathcal{L}_{i,\alpha}^V - \mathcal{L}_{j,\alpha}^V} = 1 \quad \wedge \quad \mathsf{Eval}(\mathcal{I}_\alpha, \mathcal{L}_i^V - \mathcal{L}_j^V) \cdot (R_\alpha^*)^{\mathcal{L}_{i,\alpha}^V - \mathcal{L}_{j,\alpha}^V} \neq 1 \qquad (3)$$

If we take discrete logarithms to base $\gamma \in \mathbb{G}$, where γ is an arbitrary generator of $\mathbb{G}$, and define the degree-one polynomial $\Delta_{i,j} \in \mathbb{Z}_p[X]$ as

$$\Delta_{i,j} := \log \mathsf{Eval}(\mathcal{I}_\alpha, \mathcal{L}_i^V - \mathcal{L}_j^V) + X \cdot \left(\mathcal{L}_{i,\alpha}^V - \mathcal{L}_{j,\alpha}^V\right),$$

then Equation 3 (and therefore also Equation 1) is in turn equivalent to

$$\Delta_{i,j}(\log R_\alpha) \equiv 0 \bmod p \qquad \wedge \qquad \Delta_{i,j}(\log R_\alpha^*) \not\equiv 0 \bmod p. \tag{4}$$

Similarly, Equation 2 is equivalent to

$$\Delta_{i,j}(\log R_\alpha) \not\equiv 0 \bmod p \qquad \wedge \qquad \Delta_{i,j}(\log R_\alpha^*) \equiv 0 \bmod p. \tag{5}$$

Thus, event F occurs if $\mathcal{R}$ computes vectors $\mathcal{L}_i^V, \mathcal{L}_j^V$ such that either Equation 4 or Equation 5 holds.

Failure Event F_1. Let F_1 denote the event that Equation 4 holds. Note that this can only happen if $\mathcal{R}$ performs a sequence of computations, such that there exist a pair $(i,j) \in \left[\left|\mathcal{L}^V\right|\right] \times \left[\left|\mathcal{L}^V\right|\right]$ such that the polynomial $\Delta_{i,j}$ is not the zero-polynomial in $\mathbb{Z}_p[X]$, but it holds that $\Delta_{i,j}(R_\alpha) \equiv 0 \bmod p$.

At the beginning of the game $\mathcal{R}$ receives only a random encoding $\phi(R_\alpha)$ of group element R_α. The only further information that $\mathcal{R}$ learns about R_α throughout the game is through equality or inequality of encodings. Since $\mathcal{R}$ runs in time $t_\mathcal{R}$, it can issue at most $t_\mathcal{R}$ oracle queries. Thus, at the end of the game the list $\mathcal{L}^V$ contains at most $|\mathcal{L}^V| \leq t_\mathcal{R} + q + u$ entries. Each pair $(i,j) \in \left[\left|\mathcal{L}^V\right|\right]$ with $i \neq j$ defines a (possibly non-zero) polynomial $\Delta_{i,j}$. In total there are at most $(t_\mathcal{R} + q + u) \cdot (t_\mathcal{R} + q + u - 1) \leq (t_\mathcal{R} + q + u)^2$ such polynomials.

Since all polynomials have degree one, and $\log R_\alpha$ is uniformly distributed over $\mathbb{Z}_p$ (because R_α is uniformly random over $\mathbb{G}$), the probability that $\log R_\alpha$ is a root of any of these polynomials is upper bounded by

$$\Pr[F_1] \leq \frac{(u + q + t_\mathcal{R})^2}{p}.$$

Failure Event F_2. Let F_2 denote the event that Equation 5 holds. Since $\log R_\alpha^*$ is uniformly distributed over $\mathbb{Z}_p$ (because we have defined $R_\alpha^* := g^y \mathsf{pk}^{-c}$ for uniformly $y \leftarrow\!\!\$\, \mathbb{Z}_p$), with similar arguments as before we have

$$\Pr[F_2] \leq \frac{(u + q + t_\mathcal{R})^2}{p}.$$

Bounding $\Pr[F]$. Since $F = F_1 \cup F_2$ we have

$$\Pr[F] \leq \Pr[F_1] + \Pr[F_2] \leq \frac{2(u + q + t_\mathcal{R})^2}{p}.$$

□

5 Multi-Instance Reductions

Now we turn to considering multi-instance reductions, which may run multiple sequential executions of the signature forger $\mathcal{A}$. This is the interesting case, in particular because the Forking-Lemma based security proof for Schnorr signatures by Pointcheval and Stern [25] is of this type.

Again we construct a meta-reduction with simulated adversary. The main difference to our single-instance adversary is that it does not succeed with probability 1, but tosses a biased coin that decides if it forges for the message or not. On the first glance this approach might seem to be of little value, because an adversary with a higher success probability should improve the success probability of the reduction. However, it was shown in [31] that, once we consider a reduction that runs multiple sequential executions of this adversary, this approach allows to derive an optimal tightness bound.

In the following we assume that the reduction $\mathcal{R}$ executes n sequential instances of the same adversary $\mathcal{A}(\phi(\mathsf{pk}), m, \omega)$, where the public key $\phi(\mathsf{pk})$, the message m, and the randomness ω of each instance are chosen by $\mathcal{R}$. Observe that the input to the adversary and the random oracle query/answers completely determine the behaviour of the adversary. Thus, any successive execution of an instance of $\mathcal{A}$ may be identical to a previous execution up to a certain point, where the response $c = \mathsf{H}(R, m)$ of the random oracle differs from a response $c' = \mathsf{H}(R, m)$ received by $\mathcal{A}$ in a previous execution. This point is called the *forking point* [31].

5.1 An Inefficient Adversary $\mathcal{A}$

In this section, we describe a family of inefficient forgers $\mathcal{A}$ breaking the UUF-NMA-security of the Schnorr signature scheme, as well as a specific sampling procedure to choose a random forger from this family. The specific sampling procedure guarantees that we can derive a bound on the success probability of the $\mathcal{A}$, if we sample it in the described way and then provide it to a reduction.

In the sequel, we write $\mathtt{Ber}_\mu$ to denote the Bernoulli distribution of a parameter $\mu \in [0, 1]$, i.e., $\Pr[\delta = 1] = \mu$ and $\Pr[\delta = 0] = 1 - \mu$. Let $\mathcal{Q} = \mathbb{G} \times \mathbb{Z}_p$ be the set of possible random oracle queries and answers. By $\mathbb{S}_i = \mathcal{Q}^i$ we denote the set of random oracle query sequences of length i and the set of all possible sequences is defined as $\mathbb{S} = \bigcup_{i=1}^{q} \mathbb{S}_i$. We let $\mathbb{F}$ be the set of all functions $F : \{0,1\}^k \times \mathbb{G} \times \{0,1\}^\kappa \times \mathbb{S} \to \mathbb{G}$.

1. In order to sample a forger $\mathcal{A}$, we choose a random function F uniformly from $\mathbb{F}$. Furthermore, we define a list $\Gamma \subset \mathbb{G} \times \{0,1\}$, which is generated as follows. For each (encoding of a) group element $Z \in \mathbb{G}$, list Γ contains a tuple (Z, b), where $b \leftarrow \mathtt{Ber}_\mu$. Thus, Γ contains $|\mathbb{G}|$ entries (Z, b), where each assigns a $\mathtt{Ber}_\mu$-distributed bit b to Z. For $g \in \mathbb{G}$ we write $\Gamma(Z)$ to denote the bit $b \in \{0,1\}$ such that $(Z, b) \in \Gamma$.

Note that the list Γ has exponential size, if $\mathbb{G}$ has exponential size. However, recall that we describe an *inefficient* adversary here. We will later show how this adversary can be simulated efficiently. Note furthermore that while F and Γ have been generated probabilistically, they are fixed for the given adversary, and thus can be seen as an additional, fixed "internal randomness tape" of $\mathcal{A}$.

2. The forger $\mathcal{A}$ described by F and Γ sampled as above expects as input a Schnorr public-key pk, a message m, and random coins $\omega \in \{0,1\}^\kappa$.

3. $\mathcal{A}$ sets $\sigma := \perp$ and performs the following computations. For $i = 1, \ldots, q$ it computes $R_i := F(m, \mathsf{pk}, \omega, (R_1, c_1), \ldots, (R_{i-1}, c_{i-1}))$ and queries the random oracle H on (R_i, m), where $c_i := \mathsf{H}(R_i, m) \in \mathbb{Z}_p$ is the corresponding answer. If $\sigma = \perp$, then $\mathcal{A}$ sets $Z_i := \mathsf{pk}^{c_i} R_i$ and checks if $\Gamma(Z_i) = 1$. If this is the case, then it computes $y_i \in \mathbb{Z}_p$ satisfying the equation $g^{y_i} = R_i \cdot \mathsf{pk}^{c_i}$ by exhaustive search and sets $\sigma := (R_i, y_i)$. Otherwise, if $\Gamma(Z_i) = 0$, then it continues with the loop.

4. Finally, the forger $\mathcal{A}$ returns σ.

Note that (R_i, y_i) is a valid signature for message m with respect to the public key pk. Thus, the forger $\mathcal{A}$ breaks the UUF-NMA-security of the Schnorr signatures whenever $\Gamma(Z_i) = 1$ for at least one $i \in [q]$. Hence, if we sample $\mathcal{A}$ as described above, then the probability (over the sampling and execution of $\mathcal{A}$) that it will output a valid forgery when executed by a reduction is $\epsilon_{\mathcal{A}} = 1 - (1 - \mu)^q$.

Observe that defining adversaries as above ensures that, while different instances of the same adversary will behave identically as long as their input and the answers of the random oracle are the same, as soon as one of the inputs or one of the random oracle answers differ the behavior of two instances will be independent of one another from that point onwards. As such, the behavior of these adversaries mimics closely the idea behind the forking lemma and it allows us to easily simulate the adversary in our meta-reduction below.

5.2 Main Result for Multi-Instance Reductions

In this section, we combine the approach of Seurin [31] with our simulation of signature forgeries based on re-programming of the group representation, as introduced in Section 4.1.2. This allows to prove a nearly optimal unconditional tightness bound for all generic reductions and any representation-invariant computational problem Π.

Unfortunately, the combination of the elaborate techniques of Seurin [31] with our approach yields a rather complex meta-reduction. We stress that we follow Seurin's work as closely as possible. The main difference lies in the way signature forgeries are computed, namely in our case by exploiting the properties of the generic group representation, instead of using an OMDL-oracle as in [31].

The main difference between the meta-reduction described in this section and the one presented in Section 4.1.2 lies in the simulation of the random oracle queries issued by the

adversary in different sequential executions. In particular, the meta-reduction $\mathcal{M}$ simulates the oracles procedures ENCODE, GETIDX, and $\mathcal{O}$ exactly as before.

Theorem 17. *Let Π be a representation-invariant computational problem. Suppose there exists a generic reduction $\mathcal{R}^{\mathcal{O},\mathcal{A}}$ that $(\epsilon_{\mathcal{R}}, t_{\mathcal{R}})$-solves Π, having n-time black-box access to a random instance of the hypothetical attacker described above. Then there exists an algorithm $\mathcal{M}$ that (ϵ, t)-solves Π with $t \approx t_{\mathcal{R}}$ and*

$$\epsilon \geq \epsilon_{\mathcal{R}} - \frac{2n(u + nq + t_{\mathcal{R}})}{p} - \frac{n \ln\left((1-\epsilon_{\mathcal{A}})^{-1}\right)}{q}.$$

Interpretation. Let us explain why the above theorem yields that any black-box reduction must have a security loss of $L = \Omega(q)$. First, note that a black-box reduction must work in particular for the adversaries described in Section 5.1 where $\epsilon_{\mathcal{A}}$ is a *constant* and strictly smaller than 1. In this case the theorem yields

$$\epsilon \geq \epsilon_{\mathcal{R}} - \frac{2n(u + nq + t_{\mathcal{R}})}{p} - \frac{cn}{q} \tag{6}$$

for some constant c. Note furthermore that the term $2n(u + nq + t_{\mathcal{R}})/p$ is negligible, and that the success probability ϵ of the meta-reduction must be negligible, too, if the considered representation-invariant computational problem is "hard" (which would be assumed for any meaningful reduction in a security proof). Applying this to Equation 6, we get

$$\epsilon_{\mathcal{R}} \leq \frac{c'n}{q} \tag{7}$$

for some constant c' and a sufficiently large security parameter.

To bound the loss L of any black-box reduction $\mathcal{R}$, we can now compute

$$L = \frac{t_{\mathcal{R}}}{\epsilon_{\mathcal{R}}} \cdot \frac{\epsilon_{\mathcal{A}}}{t_{\mathcal{A}}} \geq \frac{n \cdot t_{\mathcal{A}}}{\epsilon_{\mathcal{R}}} \cdot \frac{\epsilon_{\mathcal{A}}}{t_{\mathcal{A}}} = \frac{n \cdot \epsilon_{\mathcal{A}}}{\epsilon_{\mathcal{R}}} \geq \frac{q \cdot n \cdot \epsilon_{\mathcal{A}}}{c' \cdot n} = \frac{\epsilon_{\mathcal{A}}}{c'} \cdot q$$

Here, the first equality is by definition of the loss L, the first inequality uses that $\mathcal{R}$ runs the adversary n times such that we have $t_{\mathcal{R}} \geq n \cdot t_{\mathcal{A}}$, and the second inequality uses Equation 7. In conclusion, since $\epsilon_{\mathcal{A}}$ and c' are constants, we get $L = \Omega(q)$.

Proof of Theorem 17. Suppose that there exists a generic reduction $\mathcal{R} := \mathcal{R}^{\mathcal{O},\mathcal{A}}$ that $(\epsilon_{\mathcal{R}}, t_{\mathcal{R}})$-solves Π, when given access to a generic group oracle $\mathcal{O}$ and to n instances of the same forger $\mathcal{A}$, where the inputs to each instance of the forger are chosen by $\mathcal{R}$. As before, the random oracle $\mathcal{R}.\mathsf{H}$ for $\mathcal{A}$ is provided by $\mathcal{R}$. We show how to construct a meta-reduction $\mathcal{M}$ that has black-box access to $\mathcal{R}$ and that solves the representation-invariant problem Π directly. Again we proceed in a sequence of games, and denote with $\mathcal{M}_i$ the implementation of algorithm $\mathcal{M}$ in Game i, and with X_i the event that $\mathcal{R}$ outputs a valid solution $\hat{S}$ to $\hat{C}$ in

Game i. As in Section 4.1.2, we will bound the probability with which any efficient reduction $\mathcal{R}$ can distinguish each implementation $\mathcal{M}_i$ from $\mathcal{M}_{i-1}$ for all $i \in \{1, 2, 3\}$. We start with an inefficient implementation $\mathcal{M}_0$ of $\mathcal{M}$, and modify this implementation gradually until we obtain an efficient algorithm $\mathcal{M}_3$ that uses $\mathcal{R}$ to solve Π.

Game 0. $\mathcal{M}_0$ (cf. Figure 6) takes as input an instance $C = (C_1, \ldots, C_u, C') \in \mathbb{G}^u \times \{0,1\}^*$ of the representation-invariant computational problem Π and outputs a candidate solution S. It also maintains the encoding of the group using two lists $\mathcal{L}^{\mathbb{G}} \subseteq \mathbb{G}$ and $\mathcal{L}^E \subseteq E$. Our first instance $\mathcal{M}_0$ perfectly simulates one adversary chosen from the family of adversaries described above uniformly at random. The only difference between the real and the simulated adversary is that the meta-reduction does not fix the functions F and table of the adversary Γ at the beginning, but instead defines them on the fly.

Initialization of $\mathcal{M}_0$ At the beginning of the game, $\mathcal{M}_0$ chooses $\vec{R} = (R_{1,1}, \ldots, R_{n,q}) \leftarrow\!\!\$\, \mathbb{G}^{nq}$ at random (these are the values the function F will be lazily programmed to evaluate to), sets $\mathcal{I} := (C_1, \ldots, C_u, R_{1,1}, \ldots, R_{n,q})$, and runs $\textsc{Encode}(\mathcal{I})$ to assign encodings to these group elements. Furthermore, $\mathcal{M}_0$ initializes lists $\mathcal{T}$, Γ, and $\mathcal{D}$ as empty lists. Recall that $\mathcal{R}$ executes n sequential instances of the simulated adversary $\mathcal{A}$ and that, depending on the input and the query/answer pairs to $\mathcal{R}.\mathsf{H}$, the successive executions might be identical to a certain point. The list $\mathcal{T}$ will be used to store the inputs and query answer pairs of each adversary to ensure consistency of F across adversary instances. To keep the list Γ sufficiently small, it will not be fixed from the beginning, but it will be defined on-the-fly. Finally, $\mathcal{D}$ is used to store known discrete logarithms. Then, $\mathcal{M}_0$ runs a black-box simulation of the reduction $\mathcal{R}$ on input $\hat{C} := (\mathcal{L}_1^E, \ldots, \mathcal{L}_u^E, C')$. Note that $\hat{C}$ is an encoded version of the challenge instance of Π received by $\mathcal{M}_0$. That is, we have $\hat{C} = (\phi(C_1), \ldots, \phi(C_u), C')$. Oracle queries of $\mathcal{R} = \mathcal{R}^{\mathcal{O},\mathcal{A}}$ are answered exactly as described in Section 4.1.2, with the difference being the forger that we describe in the following.

The forger $\mathcal{A}(\phi(\mathsf{pk}), m, \omega)$ The simulation of the forger $\mathcal{A}$ is rather technical, because $\mathcal{M}_0$ has to provide a consistent simulation of the n sequential executions of $\mathcal{A}$. As already discussed at the beginning of this chapter, $\mathcal{M}_0$ has to emulate an identical behavior of $\mathcal{A}$ up to the forking point, or the reduction might lose its advantage. We split this algorithm up into several sub-procedures (see Figure 6). The main sub-procedures are $\textsc{BeforeFork}$ and $\textsc{AfterFork}$, with the idea that $\mathcal{A}$ runs the code of $\textsc{BeforeFork}$ if the forking point has not been reached yet and the simulation must be consistent with a previous execution. The second procedure, $\textsc{AfterFork}$ describes how $\mathcal{M}_0$ simulates $\mathcal{A}$ after the forking point.

Now we proceed with the technical description of the main procedure of $\mathcal{A}$ and explain the sub-procedures in the following. When $\mathcal{R}$ outputs $(\phi(\mathsf{pk}), m, \omega)$ to invoke an instance of $\mathcal{A}$, then $\mathcal{M}_0$'s simulation of $\mathcal{A}$ initializes the list τ with its input $(\phi(\mathsf{pk}), m, \omega)$ and the forgery σ with $\perp$. These inputs are part of the function F and we need to store them in

$\mathcal{M}_0(C)$

Initialization
parse C **as** $(C_1, \ldots, C_u, C')$
$\mathcal{L}^{\mathbb{G}} := \emptyset$
$\mathcal{L}^{E} := \emptyset$
$\vec{R} = (R_{1,1}, \ldots, R_{n,q}) \leftarrow\!\!\$\ \mathbb{G}^{nq}$
$\mathcal{I} := (C_1, \ldots, C_u, R_{1,1}, \ldots, R_{n,q})$
Encode($\mathcal{I}$)
$\mathcal{T} := \emptyset$
$\Gamma := \emptyset$
$\mathcal{D} := \emptyset$
$j := 0$
$\hat{C} := (\mathcal{L}^E_1, \ldots, \mathcal{L}^E_u, C')$
$\hat{S} \leftarrow\!\!\$\ \mathcal{R}^{\mathcal{O},\mathcal{A}}(\hat{C})$
Finalization
parse $\hat{S}$ **as** $(\hat{S_1}, \ldots, \hat{S_w}, S')$
$(i_1, \ldots, i_w) :=$ GetIdx$(\hat{S_1}, \ldots, \hat{S_w})$
return $(\mathcal{L}^{\mathbb{G}}_{i_1}, \ldots, \mathcal{L}^{\mathbb{G}}_{i_w}, S')$

$\mathcal{A}(\phi(\mathsf{pk}), m, \omega)$

$j := j + 1$
$i_{\mathsf{pk}} :=$ GetIdx$(\phi(\mathsf{pk}))$
$\tau := (\phi(\mathsf{pk}), m, \omega)$
$\sigma := \bot$
$i := 1$
BeforeFork$(m, \mathcal{L}^{\mathbb{G}}_{i_{\mathsf{pk}}})$
AfterFork$(m, \mathcal{L}^{\mathbb{G}}_{i_{\mathsf{pk}}})$
append τ **to** $\mathcal{T}$
return σ

BeforeFork(m, pk)

$k :=$ EvalF(τ)
while $k \neq \bot$ **do**
  $c_i := \mathcal{R}.\mathsf{H}(\phi(R_k), m)$
  append (k, c_i) **to** τ
  $k' :=$ EvalF(τ)
  if $\sigma = \bot$
    $Z_i := R_k \mathsf{pk}^{c_i}$
    Encode(Z_i)
    if $k' = \bot$
      Fork(Z_i, k, c_i)
      $k := k'$
    else if $\Gamma(\phi(Z_i)) = 1$
      $\sigma :=$ Forge(R_k, Z_i)
  $i := i + 1$

AfterFork(m, pk)

while $i \leq q$ **do**
  $c_i := \mathcal{R}.\mathsf{H}(\phi(R_{j,i}), m)$
  append $((j, i), c_i)$ **to** τ
  if $\sigma = \bot$
    $Z_i := R_{j,i} \mathsf{pk}^{c_i}$
    Encode(Z_i)
    if $(\phi(Z_i), b) \notin \Gamma$
      Decide$(Z_i, (j, i), c_i)$
    if $\Gamma(\phi(Z_i)) = 1$
      $\sigma :=$ Forge$(R_{j,i}, Z_i)$
    $i := i + 1$

EvalF(τ)

foreach τ' **in** $\mathcal{T}$ **do**
  if τ is a prefix of τ'
    $(k, c) := \tau'_{|\tau|+1}$
    return k
return $\bot$

Fork(Z, k, c)

if $(\phi(Z), b) \notin \Gamma$
  Decide(Z, k, c)
if $\Gamma(\phi(Z)) = 1$
  $\sigma :=$ Forge(R_k, Z)

Forge(R, Z)

foreach (Z', y') **in** $\mathcal{D}$ **do**
  if $Z' = Z$
    return $(\phi(R), y')$

Decide(Z, k, c)

$\delta_z \leftarrow\!\!\$\ \mathtt{Ber}_\mu$
$\Gamma := \Gamma \cup \{(\phi(Z), \delta_z)\}$
if $\delta_z = 1$
  $y :=$ DLog(Z, k, c)
  append (Z, y) **to** $\mathcal{D}$

DLog(Z, k, c)

foreach y **in** $\mathbb{Z}_p$ **do**
  if $g^y = Z$
    return y

Figure 6: Meta-Reduction $\mathcal{M}_0$.

order to ensure consistency with previous adversary instances.

The forger's first stage $\textbf{BeforeFork}(\mathsf{pk}, m)$: In this stage, the forger first tries to evaluate the function F on its input using EvalF. If no previous instance with the same input exists, the instance has already forked and BeforeFork immediately returns. If the instance has not yet forked from all other instances, i.e., if there exists a previous instance with the same input, it receives back the index k of the R to which F evaluates. In this case it proceeds to ask query $c_i = \mathcal{R}.\mathsf{H}(\phi(R_k), m)$ and appends (k, c_i) to τ. If it has not already forged a signature it then computes $Z_i := R_k \mathsf{pk}^{c_i}$. If the forking point has been reached, the adversary now forks from the previous instances as described in Fork. Otherwise, if $\Gamma(\phi(Z_i))$ is defined and it holds that $\Gamma(\phi(Z_i)) = 1$, then $\mathcal{A}$ forges a signature by calling $\textsc{Forge}(R_k, Z_i)$. The algorithm will repeat the described process until the forking point is reached.

The forger's second stage $\textbf{AfterFork}(\mathsf{pk}, m)$: After the current instance has forked from all previous instances it proceeds as follows. Until exactly q random oracle queries have been asked, $\mathcal{A}$ queries $c_i := \mathcal{R}.\mathsf{H}(\phi(R_{j,i}), m)$ and appends $((j, i), c_i)$ to τ. If the adversary has not already forged a signature, it continues to compute $Z_i := R_{j,i}\mathsf{pk}^{c_i}$. If $\Gamma(\phi(Z_i))$ is not yet defined, then the adversary determines the bit assigned to $\phi(Z_i)$ on-the-fly by invoking Decide. If afterwards $\Gamma(\phi(Z_i)) = 1$, then a signature is forged. The algorithm continues in this fashion until exactly q random oracle queries have been asked.

Handling the forking point $\textbf{Fork}(Z, k, c)$: When the simulation of $\mathcal{A}$ reaches the forking point, it checks whether $\Gamma(\phi(Z))$ has already been defined. If not, then the simulation calls Decide. If yes and it holds that $\Gamma(\phi(Z)) = 1$, i.e. if $\mathcal{M}$ already knows the discrete logarithm, the simulation produces a forgery.

Deciding whether to forge $\textbf{Decide}(Z, k, c)$: To assign a bit b to Z, the simulation tosses a biased coin $\delta_z \leftarrow\!\!\$\ \mathtt{Ber}_\mu$ and appends (Z, δ_z) to Γ. Furthermore, if $\delta_z = 1$ then its discrete logarithm y is computed using DLog and (Z, y) is appended to $\mathcal{D}$.

Computing the discrete logarithm $\textbf{DLog}(Z, k, c)$: Computation of the discrete logarithm is performed by exhaustively searching for a $y \in \mathbb{Z}_p$ satisfying $g^y = Z$.

Producing a forgery $\textbf{Forge}(R, Z)$: Actually producing a forgery is trivial, because forgeries will only be produced for Z with $\Gamma(Z) = 1$. By construction, for each such Z, $\mathcal{D}$ already contains the discrete logarithm. Accordingly, a forgery is produced by finding the entry $(Z', y') \in \mathcal{D}$ such that $Z' = Z$ and returning (R, y')

Finalization of $\mathcal{M}_0$ Eventually, $\mathcal{R}$ outputs a solution $\hat{S} := (\hat{S}_1, \ldots, \hat{S}_w, S') \in \hat{G}^w \times \{0,1\}^*$. Then $\mathcal{M}_0$ runs $(i_1, \ldots, i_w) := \textsc{GetIdx}(\hat{S}_1, \ldots, \hat{S}_w)$ to determine the indices of group elements $(\mathcal{L}^{\mathbb{G}}_{i_1}, \ldots, \mathcal{L}^{\mathbb{G}}_{i_w})$ corresponding to encodings $(\hat{S}_1, \ldots, \hat{S}_w)$, and outputs $(\mathcal{L}^{\mathbb{G}}_{i_1}, \ldots, \mathcal{L}^{\mathbb{G}}_{i_w}, S')$.

Analysis of $\mathcal{M}_0$ Note that $\mathcal{M}_0$ provides a perfect simulation of the oracle $\mathcal{O}$ and it also mimics the inefficient attacker from Section 5.1 perfectly, the only difference being that F is chosen lazily and that Γ is defined on-the-fly. In particular, (R, y') is a valid forgery for message m and thus, $\mathcal{R}^{\mathcal{O},\mathcal{A}}$ outputs a solution $\hat{S} = (\hat{S}_1, \ldots, \hat{S}_w, S')$ to $\hat{C}$ with probability $\Pr[X_0] = \epsilon_{\mathcal{R}}$. Since Π is assumed to be representation-invariant, $S := (S_1, \ldots, S_w, S')$ is therefore a valid solution to C, where $\hat{S}_i = \phi(S_i)$ for $i \in [w]$. Thus $\mathcal{M}_0$ outputs a valid solution S to C with probability $\epsilon_{\mathcal{R}}$.

$\mathcal{M}_1(C)$

Initialize
parse C **as** $(C_1, \ldots, C_u, C')$
$\mathcal{L}^{\mathbb{G}} := \emptyset$
$\mathcal{L}^{E} := \emptyset$
$\mathcal{L}^{V} := \emptyset$
$\vec{R} = (R_{1,1}, \ldots, R_{n,q}) \leftarrow\!\$\ \mathbb{G}^{q \cdot n}$
$\mathcal{I} := (C_1, \ldots, C_u, R_{1,1}, \ldots, R_{n,q})$
$\textsc{Encode}(\mathcal{I})$
$\mathcal{L}^{V}_i := \eta_i,\ \forall i \in [u + nq].$
$\mathcal{T} := \emptyset$
$\Gamma := \emptyset$
$\mathcal{D} := \emptyset$
$j := 0$
$\hat{C} := (\mathcal{L}^{E}_1, \ldots, \mathcal{L}^{E}_u, C')$
$\hat{S} \leftarrow\!\$\ \mathcal{R}^{\mathcal{O},\mathcal{A}}(\hat{C})$
Finalization
parse $\hat{S}$ **as** $(\hat{S}_1, \ldots, \hat{S}_w, S')$
$(i_1, \ldots, i_w) := \textsc{GetIdx}(\hat{S}_1, \ldots, \hat{S}_w)$
return $(\mathcal{L}^{\mathbb{G}}_{i_1}, \ldots, \mathcal{L}^{\mathbb{G}}_{i_w}, S')$

$\mathcal{O}(e, e', \circ)$

$(e, e', \circ) \in E \times E \times \{\cdot, \div\}$
$(i, j) := \textsc{GetIdx}(e, e')$
$a := \mathcal{L}^{V}_i \diamond \mathcal{L}^{V}_j \in \mathbb{Z}^{u+q}_p$
append a **to** $\mathcal{L}^{V}$
return $\textsc{Encode}(\mathcal{L}^{\mathbb{G}}_i \circ \mathcal{L}^{\mathbb{G}}_j)$

Figure 7: Extending $\mathcal{M}_0$ with additional bookkeeping yields $\mathcal{M}_1$. The elements highlighted in gray show the difference to $\mathcal{M}_0$. All procedures not shown are not changed.

Game 1. In this game we introduce an implementation $\mathcal{M}_1$ which extends $\mathcal{M}_0$ with bookkeeping, exactly as in Game 1 from the proof of Theorem 14. See Figure 7. Briefly summarized, we introduce an additional list $\mathcal{L}^V \subseteq \mathbb{Z}_p^{u+nq}$ to record the sequence of operations performed by $\mathcal{A}$. Let η_i denote the i-th canonical unit vector in $\mathbb{Z}_p^{u+nq}$. Then this list is initialized as $\mathcal{L}_i^V = \eta_i$ for $i \in [u + nq]$. Whenever $\mathcal{R}$ asks to perform a computation $(\mathcal{L}_i^E, \mathcal{L}_j^E, \circ)$, then $\mathcal{M}_1$ proceeds as before, but additionally appends $a := \mathcal{L}_i^V + \mathcal{L}_j^V \in \mathbb{Z}_p^{u+nq}$ (if $\circ = \cdot$) or $\mathcal{L}_i^V - \mathcal{L}_j^V \in \mathbb{Z}_p^{u+nq}$ (if $\circ = \div$) to $\mathcal{L}^V$.

Furthermore, in order to keep list $\mathcal{L}^V$ consistent with $\mathcal{L}^{\mathbb{G}}$ (exactly as in in the proof of Theorem 14), we replace the generic group oracle $\mathcal{O}$ of $\mathcal{M}_0$ with the following procedure.

Generic group oracle $\mathcal{O}(e, e', \circ)$ Given a query $(e, e', \circ) \in E \times E \times \{\cdot, \div\}$, the oracle $\mathcal{O}$ determines the smallest indices i and j such that $e = e_i$ and $e' = e_j$ by calling GetIdx. It computes $a := \mathcal{L}_i^V \diamond \mathcal{L}_j^V \in \mathbb{Z}_p^{u+nq}$, where $\diamond := +$ if $\circ = \cdot$ and $\diamond := -$ if $\circ = \div$, and appends a to $\mathcal{L}^V$. Finally it returns $\textsc{Encode}(\mathcal{L}_i^{\mathbb{G}} \circ \mathcal{L}_j^{\mathbb{G}})$.

Recall that the initial content $\mathcal{I}$ of $\mathcal{L}^{\mathbb{G}}$ is $\mathcal{I} = (C_1, \ldots, C_u, R_{1,1}, \ldots, R_{n,q})$, and that $\mathcal{R}$ performs only group operations on $\mathcal{I}$. Now, by construction of $\mathcal{M}_1$, it holds that $\mathcal{L}_i^{\mathbb{G}} = \mathsf{Eval}(\mathcal{I}, \mathcal{L}_i^V)$ for all $i \in \left[\left|\mathcal{L}^{\mathbb{G}}\right|\right]$. Thus, at any point in time during the execution of $\mathcal{R}$, the entire list $\mathcal{L}^{\mathbb{G}}$ of group elements can be recomputed from $\mathcal{L}^V$ and $\mathcal{I}$ by setting $\mathcal{L}_i^{\mathbb{G}} := \mathsf{Eval}(\mathcal{I}, \mathcal{L}_i^V)$ for $i \in \left[\left|\mathcal{L}^V\right|\right]$.

Again this change is made to keep list $\mathcal{L}^V$ consistent with $\mathcal{L}^{\mathbb{G}}$, i.e., to ensure that $\mathcal{L}_i^{\mathbb{G}} = \mathsf{Eval}(\mathcal{I}, \mathcal{L}_i^V)$ for all $i \in \left[\left|\mathcal{L}^{\mathbb{G}}\right|\right]$, where $\mathcal{I} := (C_1, \ldots, C_u, R_{1,1}, \ldots, R_{n,q})$. Clearly $\mathcal{R}$ is completely oblivious to this change, thus

$$\Pr[X_1] = \Pr[X_0]$$

$\textsc{Fork}(Z, k, c)$

if $(\phi(Z), b) \notin \Gamma$
  $\textsc{Decide}(Z, k, c)$
  if $\Gamma(\phi(Z)) = 1$
    Abort simulation
if $\Gamma(\phi(Z)) = 1$
  $\sigma := \textsc{Forge}(R_k, Z)$

Figure 8: The difference between $\mathcal{M}_1$ and $\mathcal{M}_2$.

Game 2. In this game we introduce an implementation $\mathcal{M}_2$ (cf. Figure 8) which works exactly as $\mathcal{M}_1$, except that it aborts when it would have to compute a new forgery at a forking point. That is, $\mathcal{M}_2$ aborts when it would have to forge in the case where it queried

an R_i already asked by a previous instance of the adversary but received a different answer c_i. This step is important, because in the final implementation $\mathcal{M}_3$ we will not be able to simulate valid signatures if this happens.

FORK(Z, k, c): If FORK is called on input $\phi(Z)$, such that there exists no $b \in \{0,1\}$ such that $(Z, b) \in \Gamma$, and DECIDE chooses a bit $b = 1$, then $\mathcal{M}_2$ aborts.

Analysis of $\mathcal{M}_2$. We claim that $\mathcal{R}$ is not able to distinguish $\mathcal{M}_2$ from $\mathcal{M}_1$ with probability greater than $n \ln\left((1-\epsilon_{\mathcal{A}})^{-1}\right)/q$. To show this, observe that Game 2 and Game 1 are perfectly indistinguishable, as long as $\mathcal{M}_2$ does not abort in FORK. We use Lemma 4 of [31] to bound the probability of an abort.

Lemma 18 (Based on Lemmas 1 and 4 of [31])**.** *The probability that $\mathcal{M}_2$ aborts in* FORK *is at most*

$$n\mu \leq \frac{n \ln\left((1-\epsilon_{\mathcal{A}})^{-1}\right)}{q}$$

Proof. Following [31, Lemma 4], we observe that the probability that $\mathcal{M}_2$ aborts in one execution is at most μ. With a union bound, we thus get an upper bound of $n\mu$ on the abort probability in n executions. Furthermore, by [31, Lemma 1], it holds that $q\mu \leq \ln\left((1-\epsilon_{\mathcal{A}})^{-1}\right)$, which yields the claim. □

We thus have

$$\Pr[X_2] \geq \Pr[X_1] - \Pr[F_1] \geq \Pr[X_1] - \frac{n \ln\left((1-\epsilon_{\mathcal{A}})^{-1}\right)}{q}.$$

DLOG$(Z, (j,i), c)$

$y \leftarrow\!\!\$\ \mathbb{Z}_p$
$R^*_{j,i} := g^y \cdot \mathsf{pk}^{-c}$
$(C_1, ..., C_u, R'_{1,1}, ..., R'_{q,n}) := (\mathcal{L}^{\mathbb{G}}_1, \ldots, \mathcal{L}^{\mathbb{G}}_{u+qn})$
$\mathcal{I}^* := (C_1, \ldots, C_u, R'_{1,1}, \ldots, R'_{j,i-1}, R^*_{j,i}, R'_{j,i+1}, \ldots, R'_{n,q}$
for $k = 1, \ldots, \left|\mathcal{L}^{\mathbb{G}}\right|$ **do**
  $\mathcal{L}^{\mathbb{G}}_k := \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_k)$
return y

Figure 9: The difference between $\mathcal{M}_2$ and $\mathcal{M}_3$.

Game 3. Note that the meta-reductions described in previous games were not efficient, because the simulation of the attacker in procedure $\mathcal{A}$ needed to compute a discrete logarithm by exhaustive search. In this final game, we construct an efficient meta-reduction

$\mathcal{M}_3$ that it identical to $\mathcal{M}_2$, with the difference that it simulates $\mathcal{A}$ efficiently. $\mathcal{M}_3$ proceeds exactly like $\mathcal{M}_2$, except for the following (cf. Figure 9).

DLog(Z, k, c): The DLog procedure chooses $y \leftarrow\!\!\$\, \mathbb{Z}_p$ uniformly random and computes

$$R^*_{j,i} := g^y \cdot \mathsf{pk}^{-c} \tag{8}$$

Then it reads the first $u + qn$ entries from $\mathcal{L}^{\mathbb{G}}$ as

$$(C_1, ..., C_u, R'_{1,1}, ..., R'_{q,n}) := (\mathcal{L}^{\mathbb{G}}_1, \ldots, \mathcal{L}^{\mathbb{G}}_{u+qn}),$$

replaces $R_{j,i}$ with $R^*_{j,i}$ by setting

$$\mathcal{I}^* := (C_1, \ldots, C_u, R'_{1,1}, \ldots, R'_{j,i-1}, R^*_{i,j}, R'_{j,i+1}, \ldots, R'_{q,n}),$$

and finally re-computes the entire list $\mathcal{L}^{\mathbb{G}}$ from $\mathcal{L}^V$ by setting $\mathcal{L}^{\mathbb{G}}_a := \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_a)$ for all $a \in \left[\left|\mathcal{L}^V\right|\right]$. Note that this implicitly defines Z as $Z := g^y$, due to Equation 8.

Note that meta-reduction $\mathcal{M}_3$ can be implemented efficiently, as it does not have to compute discrete logarithms. It remains to show that it is indistinguishable from $\mathcal{M}_2$ for $\mathcal{R}$ with all but negligible probablility.

Analysis of $\mathcal{M}_3$. First note that each σ with $\sigma \neq \perp$ output by $\mathcal{A}$ is a valid signature. Moreover, we claim that $\mathcal{R}$ is not able to distinguish $\mathcal{M}_3$ from $\mathcal{M}_2$, except for a negligibly small probability. To this end, we apply a lemma which is very similar to Lemma 16 from the proof of Theorem 14.

Lemma 19. *Let F_2 denote the event that $\mathcal{R}$ computes vectors $\mathcal{L}^V_a, \mathcal{L}^V_b \in \mathcal{L}^V$ such that*

$$\begin{aligned} &\mathsf{Eval}(\mathcal{I}, \mathcal{L}^V_a) = \mathsf{Eval}(\mathcal{I}, \mathcal{L}^V_b) \quad \wedge \quad \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_a) \neq \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_b) \\ &\qquad\qquad\qquad\qquad \textit{or} \\ &\mathsf{Eval}(\mathcal{I}, \mathcal{L}^V_a) \neq \mathsf{Eval}(\mathcal{I}, \mathcal{L}^V_b) \quad \wedge \quad \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_a) = \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_b). \end{aligned}$$

Then

$$\Pr[F_2] \leq \frac{2n(u + nq + t_{\mathcal{R}})}{p}.$$

Before we sketch the proof of this lemma (which is very similar to the proof of Lemma 16), let us finish the proof of Theorem 14. Note that $\mathcal{M}_3$ is perfectly indistinguishable from $\mathcal{M}_2$, unless Event F occurs. Applying the above lemma, we thus obtain

$$\Pr[X_3] \geq \Pr[X_2] - \Pr[F_2] \geq \Pr[X_2] - \frac{2n(u + nq + t_{\mathcal{R}})}{p}.$$

Summing up, we thus obtain that

$$\epsilon \geq \epsilon_{\mathcal{R}} - \frac{2n(u+nq+t_{\mathcal{R}})}{p} - \frac{n \ln\left((1-\epsilon_{\mathcal{A}})^{-1}\right)}{q}.$$

□

Proof Sketch for Lemma 19. The proof of Lemma 19 is almost identical to the proof of Lemma 16. The main difference is that we need to simulate many (up to n) signatures in the multi-instance case. This works well, with the same arguments as in the proof of Lemma 16, as long as we make sure that we do not need to re-assign the same encoding twice. (In particular because this would invalidate a signature previously computed by $\mathcal{A}$, and thus be easily noticeable for $\mathcal{R}$.)

By construction of $\mathcal{M}_3$, this can happen only if FORK receives as input a group element Z such that $\Gamma(\phi(Z)) = 1$. Note that this is exactly when event F_1 occurs, in which case the game is aborted anyway, due to the changes introduced in Game 2.

Suppose that event F_1 does not occur. In this case we re-assign each encoding at most once, by replacing in list $\mathcal{L}^{\mathbb{G}}$ a uniformly distributed group element $R_{i,j}$ with another uniform group element $R^*_{i,j}$, and re-computing all group elements contained in $\mathcal{L}^{\mathbb{G}}$. Following Lemma 16, each replacement can be noticed by $\mathcal{R}$ with probability at most

$$\frac{2(u+nq+t_{\mathcal{R}})}{p},$$

where the term $u+nq$ (instead of $u+q$ as before) is due to the fact that in the multi-instance case $\mathcal{L}^{\mathbb{G}}$ is now initialized with $u+nq$ group elements. Since in total at most n encodings are re-assigned throughout the game, a union bound yields

$$\Pr[F_2] \leq \frac{2n(u+nq+t_{\mathcal{R}})}{p}.$$

□

6 On the Existence of Generic Reductions in the NPROM

The non-programmable random oracle model (NPROM) was first introduced by Fischlin et al. [15] as a realaxation of the random oracle model. In this relaxation, the random oracle is a uniformly chosen random function that is external to the reduction, meaning that the reduction can no longer reprogram the oracle's answers. The model does however preserve the observability property of the ROM, i.e., the reduction can still observe all the queries made by the adversary. As such, the NPROM remains an idealized model but a weaker one.

In this section we apply our meta-reduction technique to investigate the possibility of finding any generic reduction $\mathcal{R}$ (not just tight ones) that reduces a representation-invariant computational problem Π to breaking the UUF-NMA-security of the Schnorr signature scheme in the, weaker, *Non-Programmable* Random Oracle Model. This question orthogonal to the search for tight security proofs in the random oracle model. As in the sections before, our results here are negative. We prove that it is impossible to find a generic reduction from any non-interactive representation-invariant computational problem in the NPROM.

6.1 An Inefficient adversary $\mathcal{A}$

We once again describe an inefficient adversary $\mathcal{A}$ that breaks UUF-NMA-security of Schnorr signatures. The adversary $\mathcal{A}$ for this case is very simple.

1. The input of $\mathcal{A}$ is a Schnorr public key $\mathsf{pk} \in \mathbb{G}$, a message $m \in \{0,1\}^k$, and random coins $\omega \in \{0,1\}^\kappa$.
2. The forger $\mathcal{A}$ chooses a uniformly random $R \leftarrow_\$ \mathbb{G}$, queries the random oracle to compute $c := \mathsf{H}(R, m)$ and computes $Z := \mathsf{pk}^c R$.
3. Finally, the forger $\mathcal{A}$ uses exhaustive search to find $y \in \mathbb{Z}_p$ such that $Z = g^y$ and outputs (R, y).

Note that (R, y) is by definition of the Schnorr signature scheme always a valid signature for message m under public key pk. Thus, the forger described above breaks the UUF-NMA-security of Schnorr signatures with probability 1.

6.2 Main Result for Reductions in the NPROM

We will prove the following Theorem.

Theorem 20. *Let $\Pi = (\mathcal{G}_\Pi, \mathcal{V}_\Pi)$ be a representation-invariant non-interactive computational problem. Suppose there exists a generic reduction $\mathcal{R}$ that $(\epsilon_\mathcal{R}, t_\mathcal{R})$-solves Π, having n-*time *black-box access to the hypothetical attacker $\mathcal{A}$ described above in the non-programmable random oracle model. Then there exists an algorithm $\mathcal{M}$ that (ϵ, t)-solves Π with $t \approx t_\mathcal{R}$ and*

$$\epsilon \geq \epsilon_\mathcal{R}(1 - 2n(u + t_\mathcal{R})/p).$$

Remark 21. The values n, u, and $t_\mathcal{R}$ are polynomially bounded while p is exponential. Therefore, the theorem shows that the existence of a reduction $\mathcal{R}$ implies the existence of a meta-reduction $\mathcal{M}$, which solves Π with essentially the same success probability and running time. Thus, an efficient (and even *non*-tight) reduction $\mathcal{R}$ can only exist if there exists an efficient algorithm for Π, which means that Π cannot be hard.

$\mathcal{M}(C)$	$\mathcal{A}(\phi(\mathsf{pk}), m, \omega)$
# INITIALIZATION	$e \leftarrow\!\!\$\, E$
parse C **as** $(C_1, \ldots, C_u, C')$	**if** $e \in \mathcal{L}^E$
$\mathcal{L}^{\mathbb{G}} := \emptyset$	**Abort simulation**
$\mathcal{L}^E := \emptyset$	$c := \mathcal{R}.\mathsf{H}(e, m)$
ENCODE$(C_1, \ldots, C_u)$	$y \leftarrow\!\!\$\, \mathbb{Z}_p$
$\hat{C} := (\mathcal{L}_1^E, \ldots, \mathcal{L}_u^E, C')$	$i :=$ GETIDX$(\phi(\mathsf{pk}))$
$\hat{S} \leftarrow\!\!\$\, \mathcal{R}^{\mathcal{O},\mathcal{A}}(\hat{C})$	$R := g^y \cdot (\mathcal{L}_i^{\mathbb{G}})^{-c}$
# FINALIZATION	**if** $R \in \mathcal{L}^{\mathbb{G}}$
parse $\hat{S}$ **as** $(\hat{S}_1, \ldots, \hat{S}_w, S')$	**Abort simulation**
$(i_1, \ldots, i_w) :=$ GETIDX$(\hat{S}_1, \ldots, \hat{S}_w)$	**append** e **to** $\mathcal{L}^E$
return $(\mathcal{L}_{i_1}^{\mathbb{G}}, \ldots, \mathcal{L}_{i_w}^{\mathbb{G}}, S')$	**append** R **to** $\mathcal{L}^{\mathbb{G}}$
	return (e, y)

Figure 10: Implementation of $\mathcal{M}$.

Proof. Assume that there exists a generic reduction $\mathcal{R} := \mathcal{R}^{\mathcal{O},\mathcal{A}}$ that $(\epsilon_{\mathcal{R}}, t_{\mathcal{R}})$-solves Π when given access to a generic group oracle $\mathcal{O}$, and a forger $\mathcal{A}(\phi(\mathsf{pk}), m, \omega)$, where the inputs to the forger are chosen by $\mathcal{R}$. Furthermore, the reduction $\mathcal{R}$ can observe all random oracle queries made by $\mathcal{A}$, however it cannot influence the responses. We show how to build a meta-reduction $\mathcal{M}$ that has black-box access to $\mathcal{R}$ and solves the representation-invariant problem Π directly.

Note that $\mathcal{R}$ cannot influence the random oracle responses, but only the initial inputs to the adversary. If the reduction runs $\mathcal{A}$ twice on the same initial inputs, then all queries made by $\mathcal{A}$ and its output will be identical. Hence, we may assume without loss of generality that $\mathcal{R}$ will never invoke $\mathcal{A}$ on the same input twice. This makes things much simpler, as we do not have to ensure consistency between different instances of the adversary.

Meta-reduction $\mathcal{M}$. At the beginning of the game, $\mathcal{M}$ receives a challenge $C = (C_1, \ldots, C_u, C')$. It initializes the lists $\mathcal{L}^{\mathbb{G}} := \emptyset$ and $\mathcal{L}^E := \emptyset$ and determines encodings by running $(\phi(C_1), \ldots, \phi(C_u)) =$ ENCODE$(C_1, \ldots, C_u)$. Then it invokes $\mathcal{R}^{\mathcal{A},\mathcal{O}}(\phi(C_1), \ldots, \phi(C_u), C')$. The oracle $\mathcal{O}$ is simulated exactly as in previous proofs.

Whenever $\mathcal{R}$ outputs $(\phi(\mathsf{pk}), m, \omega)$ to invoke an instance of $\mathcal{A}$, $\mathcal{M}$ proceeds as follows. It chooses a random encoding $e \leftarrow\!\!\$\, E$, and raises event F_1 and aborts if $e \in \mathcal{L}^E$. Then it queries the random oracle provided by $\mathcal{R}$ to compute $c := \mathcal{R}.\mathsf{H}(e, m)$, chooses $y \leftarrow\!\!\$\, \mathbb{Z}_p$, calls $i :=$ GETIDX$(\phi(\mathsf{pk}))$, and computes $R := g^y \cdot (\mathcal{L}_i^{\mathbb{G}})^{-c}$. It raises event F_2 and aborts if $R \in \mathcal{L}^{\mathbb{G}}$. Finally $\mathcal{M}$ appends e to $\mathcal{L}^E$ and R to $\mathcal{L}^{\mathbb{G}}$ and outputs (e, y) as a forgery.

Eventually, the algorithm $\mathcal{R}$ outputs a solution $\hat{S} := (\hat{S}_1, \ldots, \hat{S}_w, S') \in E^w \times \{0,1\}^*$.

The algorithm $\mathcal{M}$ runs $(i_1, \ldots, i_w) := \textsc{GetIdx}(\hat{S}_1, \ldots, \hat{S}_w)$ to determine the indices of group elements $(\mathcal{L}^{\mathbb{G}}_{i_1}, \ldots, \mathcal{L}^{\mathbb{G}}_{i_w})$ corresponding to encodings $(\hat{S}_1, \ldots, \hat{S}_w)$, and outputs $(\mathcal{L}^{\mathbb{G}}_{i_1}, \ldots, \mathcal{L}^{\mathbb{G}}_{i_w}, S')$.

Analysis of $\mathcal{M}$. Note that $\mathcal{M}$ provides a perfect simulation of the oracle $\mathcal{O}$. Further, it mimics the attacker from Section 6.1 perfectly unless it aborts while attempting to simulate a forger. In particular, (R, y) is always a valid forgery for message m and thus, $\mathcal{R}$ outputs a solution $\hat{S} = (\hat{S}_1, \ldots, \hat{S}_w, S')$ to $\hat{C}$ with probability $\epsilon_{\mathcal{R}}$. Since Π is assumed to be representation-invariant, $S := (S_1, \ldots, S_w, S')$ with $\hat{S}_i = \phi(S_i)$ for $i \in [w]$ is therefore a valid solution to C. Therefore, the success probability of $\mathcal{M}$ is at least

$$\epsilon \geq \epsilon_{\mathcal{R}} \cdot (1 - \Pr[F_1 \cup F_2]).$$

Since the n encodings chosen while simulating the forger $\mathcal{A}$ are chosen uniformly at random, and we have $\left|\mathcal{L}^E\right| \leq u + t_{\mathcal{R}}$ at all times, we can bound the probability that F_1 occurs using a union bound as $\Pr[F_1] \leq n \cdot (u + t_{\mathcal{R}})/p$. Similarly, since $y \leftarrow\!\!\$\, \mathbb{Z}_p$ is uniformly random and therefore R is a uniformly random group element in each simulated forger, we have $\Pr[F_2] \leq n \cdot (u + t_{\mathcal{R}})/p$. Therefore, using another union bound, we get that $\Pr[F_1 \cup F_2] \leq \Pr[F_1] + \Pr[F_2] \leq 2n(u + t_{\mathcal{R}})/p$.

Thus, in conclusion we obtain that

$$\epsilon \geq \epsilon_{\mathcal{R}} \left(1 - \frac{2n(u + t_{\mathcal{R}})}{p}\right)$$

as claimed. □

Remark 22. Note that in the NPROM the reduction is no longer able to program the random oracle, which significantly reduces its the power and makes it easier to simulate the hypothetical adversary. Concretely, in the NPROM we have a leverage to argue that the probability of event F_2 is negligible, which is not possible in the ROM and significantly simplifies the meta-reduction.

It is interesting to note that the above result does not carry over to the standard model. The reason is that – as mentioned before – the adversary is not necessarily generic. In the standard model, however, the hash function must be evaluated locally by both the reduction and the adversary. Since they are using different encodings of the group elements, a signature that appears valid for the adversary is invalid from the point of view of the reduction with overwhelming probability. One might attempt to rectify this by specifying different hash functions to adversary and reduction, i.e., specifying $\mathsf{H}(\phi(\cdot))$ as the hash function for the adversary. However, this fails for the simple reason that $\phi(\cdot)$ is not necessarily efficiently computable.

Acknowledgements. We thank the anonymous reviewers of ASIACRYPT 2014 and the Journal of Cryptology for their helpful comments, which helped us to improve the paper significantly.

References

[1] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In David Naccache, editor, *Topics in Cryptology – CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 143–158, San Francisco, CA, USA, April 8–12, 2001. Springer, Heidelberg, Germany.

[2] Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. *Journal of Cryptology*, 16(3):185–215, June 2003.

[3] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press.

[4] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 399–416, Saragossa, Spain, May 12–16, 1996. Springer, Heidelberg, Germany.

[5] Daniel J. Bernstein. Proving tight security for Rabin-Williams signatures. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 70–87, Istanbul, Turkey, April 13–17, 2008. Springer, Heidelberg, Germany.

[6] Daniel J. Bernstein. Multi-user schnorr security, revisited. Cryptology ePrint Archive, Report 2015/996, 2015. https://eprint.iacr.org/2015/996.

[7] Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 443–459, Santa Barbara, CA, USA, August 15–19, 2004. Springer, Heidelberg, Germany.

[8] Jean-Sébastien Coron. On the exact security of full domain hash. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 229–235, Santa Barbara, CA, USA, August 20–24, 2000. Springer, Heidelberg, Germany.

[9] Jean-Sébastien Coron. Optimal security proofs for PSS and other signature schemes. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 272–287, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Heidelberg, Germany.

[10] Yevgeniy Dodis, Iftach Haitner, and Aris Tentes. On the instantiability of hash-and-sign RSA signatures. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 112–132, Taormina, Sicily, Italy, March 19–21, 2012. Springer, Heidelberg, Germany.

[11] Yevgeniy Dodis, Roberto Oliveira, and Krzysztof Pietrzak. On the generic insecurity of the full domain hash. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 449–466, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Heidelberg, Germany.

[12] Yevgeniy Dodis and Leonid Reyzin. On the power of claw-free permutations. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN 02: 3rd International Conference on Security in Communication Networks*, volume 2576 of *Lecture Notes in Computer Science*, pages 55–73, Amalfi, Italy, September 12–13, 2003. Springer, Heidelberg, Germany.

[13] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany.

[14] Marc Fischlin and Nils Fleischhacker. Limitations of the meta-reduction technique: The case of Schnorr signatures. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 444–460, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany.

[15] Marc Fischlin, Anja Lehmann, Thomas Ristenpart, Thomas Shrimpton, Martijn Stam, and Stefano Tessaro. Random oracles with(out) programmability. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 303–320, Singapore, December 5–9, 2010. Springer, Heidelberg, Germany.

[16] Nils Fleischhacker, Tibor Jager, and Dominique Schröder. On tight security proofs for Schnorr signatures. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 512–531, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014. Springer, Heidelberg, Germany.

[17] Steven D. Galbraith, John Malone-Lee, and Nigel P. Smart. Public key signatures in the multi-user setting. *Inf. Process. Lett.*, 83(5):263–266, 2002.

[18] Sanjam Garg, Raghav Bhaskar, and Satyanarayana V. Lokam. Improved bounds on security reductions for discrete log based signatures. In David Wagner, editor, *Advances*

in Cryptology – CRYPTO 2008, volume 5157 of *Lecture Notes in Computer Science*, pages 93–107, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Heidelberg, Germany.

[19] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[20] Saqib A. Kakvi and Eike Kiltz. Optimal security proofs for full domain hash, revisited. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 537–553, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

[21] Eike Kiltz, Daniel Masny, and Jiaxin Pan. Optimal security proofs for signatures from identification schemes. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 33–61, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

[22] Ueli M. Maurer. Abstract models of computation in cryptography (invited paper). In Nigel P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *Lecture Notes in Computer Science*, pages 1–12, Cirencester, UK, December 19–21, 2005. Springer, Heidelberg, Germany.

[23] Gregory Neven, Nigel Smart, and Bogdan Warinschi. Hash function requirements for Schnorr signatures. *Journal of Mathematical Cryptology*, 3(1):69–87, 2009.

[24] Pascal Paillier and Damien Vergnaud. Discrete-log-based signatures may not be equivalent to discrete log. In Bimal K. Roy, editor, *Advances in Cryptology – ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 1–20, Chennai, India, December 4–8, 2005. Springer, Heidelberg, Germany.

[25] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398, Saragossa, Spain, May 12–16, 1996. Springer, Heidelberg, Germany.

[26] Omer Reingold, Luca Trevisan, and Salil P. Vadhan. Notions of reducibility between cryptographic primitives. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 1–20, Cambridge, MA, USA, February 19–21, 2004. Springer, Heidelberg, Germany.

[27] Andy Rupp, Gregor Leander, Endre Bangerter, Alexander W. Dent, and Ahmad-Reza Sadeghi. Sufficient conditions for intractability over black-box groups: Generic lower

bounds for generalized DL and DH problems. In Josef Pieprzyk, editor, *Advances in Cryptology – ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 489–505, Melbourne, Australia, December 7–11, 2008. Springer, Heidelberg, Germany.

[28] Sven Schäge. Tight proofs for signature schemes without random oracles. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 189–206, Tallinn, Estonia, May 15–19, 2011. Springer, Heidelberg, Germany.

[29] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany.

[30] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, January 1991.

[31] Yannick Seurin. On the exact security of Schnorr-type signatures in the random oracle model. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 554–571, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

[32] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266, Konstanz, Germany, May 11–15, 1997. Springer, Heidelberg, Germany.

[33] Brent R. Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 114–127, Aarhus, Denmark, May 22–26, 2005. Springer, Heidelberg, Germany.