On Tight Security Proofs for Schnorr Signatures

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

[1] Saarland University, Germany
[2] Horst Görtz Institute for IT Security
Ruhr-University Bochum, Germany

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. 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 and most reductions in state-of-the-art security proofs have this desirable property. Our approach shows *unconditionally* that there is no tight generic reduction from any *natural* computational problem Π defined over algebraic groups (including even interactive problems) to breaking Schnorr signatures, unless solving Π is easy.

Keywords: Schnorr signatures, black-box reductions, generic reductions, algebraic reductions, tightness.

1 Introduction

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 a very desirable property of a cryptosystem to have a tight security reduction.

[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.

In the domain of digital signatures tight reductions are known for many fundamental schemes, like Rabin/Williams signatures (Bernstein, Eurocrypt 2008 [5]), many strong-RSA-based signatures (Schäge, Eurocrypt 2011 [25]), and RSA Full-Domain Hash (Kakvi and Kiltz, Eurocrypt 2012 [18]). The Schnorr signature scheme [26, 27] is one of the most fundamental public-key cryptosystems. Pointcheval and Stern have shown that Schnorr signatures are provably secure, assuming the hardness of the discrete logarithm (DL) problem [22], in the Random Oracle Model (ROM) [3]. However, the reduction of Pointcheval and Stern from DL 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.

A long line of research investigates the existence of tight security proofs for Schnorr signatures. At Asiacrypt 2005 Paillier and Vergnaud [21] gave a first lower bound showing that any algebraic reduction (even in the ROM) converting a forger for Schnorr signatures into an algorithm solving some computational problem Π 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 (UF-NM, see Section A 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.* [15] 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 [28] finally closed the gap between the security proof of [22] and known impossibility results, by describing an elaborate simulation strategy for the forger and providing a new analysis. All previous works [21, 15, 28] 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 certain strong computational problems, namely the (one-more) discrete logarithm problem [1]. Reduction from weaker problems like, e.g., the computational or decisional Diffie-Hellman problem (CDH/DDH) are not considered.
2. The impossibility results 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 unlikely that first the inexistence 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 [18] 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. As all previous works rule out only tight reductions from strong computational problems like 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 based on any weaker computational problem exist?*

Our contribution. In this work we answer this question in the negative ruling out the existence of tight reductions in the random oracle model for virtually all natural computational problems defined over abstract algebraic groups. Like previous works, we consider universal unforgeability under no-message attacks (UF-NM-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 [21] applies equally to generic reductions. In particular, to the best of our knowledge all known examples of algebraic reductions are 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 [21, 15, 28] and which allows us to show the following main result:

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

Technical approach. We begin with the hypothesis that there exists a tight generic reduction $\mathcal{R}$ from some hard (and possibly interactive) problem Π to the UF-NM-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 $\mathcal{R}$.

All previous works in this direction [21, 15, 28] 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 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 a new leverage to simulate a successful forger efficiently, essentially by suitably re-programming the group representation to compute valid signatures. The technical challenge is to prove that the reduction does not notice that the meta-reduction changes 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 [30], which originally was 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 (even *interactive*) *representation-invariant* computational problem. Since almost all common hardness assumptions in algebraic groups (e.g., DL, CDH, DDH, OMDL, 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 assumption. 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 a particular representation of group elements. A generic reduction, however, is required to work essentially identical for any representation of group elements. Generic reductions are by definition more restrictive than algebraic ones, however, 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 [21, 15, 28]. 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 re-prove 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. One might wonder whether our result is implied by previous works (in particular by [28]), since we are considering generic reductions, because for generic algorithms most non-trivial computational problems in algebraic groups are equivalent to the discrete logarithm problem. The conclusion that therefore our result is implied by previous works is however not correct.

Note that a reduction does not solve the computational problem alone. It has access to an attacker $\mathcal{A}$. The algorithm which solves the computational problem is a composition $\mathcal{R}(\mathcal{A})$ of $\mathcal{R}$ and $\mathcal{A}$. If both $\mathcal{R}$ and $\mathcal{A}$ were generic algorithms, then the composition $\mathcal{R}(\mathcal{A})$ would also be a generic algorithm, and thus our results would indeed be trivial. But note that we do not require $\mathcal{A}$ to be generic. Therefore also the composition $\mathcal{R}(\mathcal{A})$

is not a generic algorithm, thus the generic equivalence of DLOG and other problems does not apply. See Section 2.4 and Figure 2 for further explanation.

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 [18].

In the context of Schnorr signatures, Neven *et al.* [20] described necessary conditions the hash function must meet in order to provide existential unforgeability under chosen-message attacks (EUF-CM), and showed that these conditions are sufficient if the forger (not the reduction!) is modeled as a generic group algorithm.

In [13] Fischlin and Fleischhacker presented a result also about the security of Schnorr signatures which is orthogonal to our result. They show, again under the OMDL assumption, that a large class of reductions has to rely on re-programming the random oracle. Essentially they prove that in the *non-programmable* ROM [14] no reduction from the discrete logarithm problem can exist that invokes the adversary only ever 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. As said before, the result is orthogonal to our main result, as it considers reductions in the *non-programmable* ROM.

2 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., $[n] := \{1, \ldots, n\}$.

2.1 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 [26, 27] consists of the following efficient algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$.

$\mathsf{Gen}(g)$**:** The key generation algorithm takes as input a generator g of $\mathbb{G}$. It chooses $x \leftarrow_\$ \mathbb{Z}_p$, computes $X := g^x$, and outputs (X, x).

$\mathsf{Sign}(x, m)$**:** The input of the signing algorithm is a private key x 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 $c := H(R, m)$, and computes $y := x \cdot c + r \bmod p$.

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

Remark 1. Note that the above description of Schnorr signatures deviates slightly from the original description in [26, 27], where a signature consists of (c, y) instead of (R, y), which reduces the length of signatures significantly. However, note that it is possible to compute R from (c, y) as $R := g^y \cdot X^{-c}$. Similarly, it is possible to compute c from (R, m) as $c := H(R, m)$. Thus both representations are equivalent. In particular, changing between these two representation does not affect our results.

2.2 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 $\mathsf{desc}(\mathbb{G}, g) = (g, g^2, \dots, g^p) \in \mathbb{G}^p$. We say that $\mathsf{desc}(\mathbb{G}, g)$ is the *enumerating description* of $\mathbb{G}$ with respect to g.

Definition 1. *A* computational problem *Π in $\mathbb{G}$ is specified by three (computationally unbounded) procedures $\Pi = (\mathcal{G}_\Pi, \mathcal{S}_\Pi, \mathcal{V}_\Pi)$, 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) $C = (C_1, \dots, C_u, C') \in \mathbb{G}^u \times \{0,1\}^*$. We assume in the sequel that at least C_1 is a generator of $\mathbb{G}$.*

$\mathcal{S}_\Pi(\mathsf{desc}(\mathbb{G}, g), st, Q)$ *takes as input* $\mathsf{desc}(\mathbb{G}, g)$*, a state st, and $Q = (Q_1, \dots, Q_v, Q') \in \mathbb{G}^v \times \{0,1\}^*$, and outputs (st', A) where st' is an updated state and $A = (A_1, \dots, A_\nu, A') \in \mathbb{G}^\nu \times \{0,1\}^*$.*

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

If $\mathcal{S}_\Pi$ always responds with $A = \emptyset$ (i.e., the empty string), then we say that Π is non-interactive. *Otherwise it is* interactive. *The exact description and distribution of st, C, Q, A, S depends on the considered computational problem.*

Definition 2. *An algorithm $\mathcal{A}$ (ϵ, t)-solves the 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 most 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 sends C to $\mathcal{A}$.*
2. *$\mathcal{A}$ is allowed to issue an arbitrary number of* oracle queries *to $\mathcal{C}$. To this end, $\mathcal{A}$ provides $\mathcal{C}$ with a query Q. $\mathcal{C}$ runs $(st', A) \leftarrow_\$ \mathcal{S}_\Pi(\mathsf{desc}(\mathbb{G}, g), st, Q)$, updates the state $st := st'$, and responds with A.*
3. *Finally, algorithm $\mathcal{A}$ outputs a candidate solution S. The algorithm $\mathcal{A}$ wins the game (i.e., solves the computational problem correctly) iff $\mathcal{V}_\Pi(\mathsf{desc}(\mathbb{G}, g), st, C, S) = 1$.*

Example 1. 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. $\mathcal{S}_\Pi(\mathsf{desc}(\mathbb{G}, g), st, Q)$ always outputs $(st', A) = (st, \emptyset)$. $\mathcal{V}_\Pi(\mathsf{desc}(\mathbb{G}, g), st, C, S)$ interprets $S = S' \in \{0,1\}^*$ canonically as an integer in $\mathbb{Z}_p$, and outputs 1 iff $h = g^{S'}$.

Example 2. We describe the *u-one-more discrete logarithm problem* (u-OMDL) [2, 1] in $\mathbb{G}$ with the following algorithms. $\mathcal{G}_\Pi(\mathsf{desc}(\mathbb{G}, g))$ outputs (st, C) where $C = (C_1, \dots, C_u) \leftarrow_\$ \mathbb{G}^u$ consists of u random group elements and $st = 0$. The algorithm

$\mathcal{S}_\Pi(\mathsf{desc}(\mathbb{G}, g), st, Q)$ takes as input state st and group element $Q \in \mathbb{G}$. It responds with $st' := st + 1$ and $A = A' \in \{0,1\}^*$, where A' canonically interpreted as an integer in $\mathbb{Z}_p$ satisfies $g^{A'} = Q$. The verification algorithm $\mathcal{V}_\Pi(\mathsf{desc}(\mathbb{G}, g), st, C, S)$ interprets $S = (S'_1, \dots, S'_u) \in \{0,1\}^*$ canonically as a vector of u integers in $\mathbb{Z}_p$, and outputs 1 iff $st < u$ and $g_i = g^{S'_i}$ for all $i \in [u]$.

Example 3. The UF-NM-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 $C = (g, X, m) \in \mathbb{G}^2 \times \{0,1\}^k$, where $X = g^x$ for $x \leftarrow_\$ \mathbb{Z}_p$ and $m \leftarrow_\$ \{0,1\}^k$. $\mathcal{S}_\Pi(\mathsf{desc}(\mathbb{G}, g), st, Q)$ always outputs $(st', A) = (st, \emptyset)$. 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 $c := H(R, st)$, and outputs 1 iff $X^c \cdot R = g^y$.

2.3 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 1. However, it is easy to see that this is not achievable in this generality: as Example 3 shows, the problem of forging Schnorr signatures itself is a problem that meets Definition 1. However, of course 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, even interactive assumptions, except for a few extremely artificial problems. In particular, the problem of forging Schnorr signatures is *not* contained in this class (see Example 5 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, oracle queries, and solutions is converted to a different representation of the same group. More formal is the following definition:

Definition 3. *Let $\mathbb{G}, \hat{\mathbb{G}}$ be groups such that there exists an isomorphism $\phi : \mathbb{G} \to \hat{\mathbb{G}}$. We say that Π is* representation-invariant, *if for all isomorphic groups $\mathbb{G}, \hat{\mathbb{G}}$ and for all generators $g \in \mathbb{G}$, all $C = (C_1, \dots, C_u, C') \leftarrow_\$ \mathcal{G}_\Pi(\mathsf{desc}(\mathbb{G}, g))$, all $st = (st_1, \dots, st_t, st') \in \mathbb{G}^t \times \{0,1\}^*$, and all $S = (S_1, \dots, S_w, S') \in \mathbb{G}^w \times \{0,1\}^*$ 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 $\hat{g} = \phi(g) \in \mathbb{G}'$, $\hat{C} = (\phi(C_1), \dots, \phi(C_u), C')$, $\hat{st} = (\phi(st_1), \dots, \phi(st_t), st')$, and $\hat{S} = (\phi(S_1), \dots, \phi(S_w), S')$.*

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 4. 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), one-more discrete log (OMDL), decision linear (DLIN), and so on.

Example 5. The UF-NM-forgery problem for Schnorr signatures with hash function H is *not* representation-invariant for any hash function H. Let $C = (g, X, m) \leftarrow_\$ \mathcal{G}_\Pi(\mathsf{desc}(\mathbb{G}, g))$ be a challenge with solution $S = (R, y) \in \mathbb{G} \times \mathbb{Z}_p$ satisfying $X^c \cdot R = g^y$, where $c := H(R, m)$.

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 $H(R, m) \neq H(\phi(R), m)$ in $\mathbb{Z}_p$ (which holds in particular if H is collision resistant), then we have

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

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

The UF-NM-forgery problem of Schnorr signatures is not representation-invariant, because a solution to this problem involves the hash value $H(R, m)$ 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.

2.4 Generic Reductions

In this section we recall the notion of *generic groups*, loosely following [30] (cf. also [19, 24], 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 [30] 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 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 $S' \in \{0,1\}^*$

[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.

```
PROC $\mathcal{O}(e, e', \circ)$
$(e, e', \circ) \in E \times E \times \{\cdot, \div\}$
$(i, j) :=$ GETIDX$(e, e')$
return ENCODE$(\mathcal{L}_i^{\mathbb{G}} \circ \mathcal{L}_j^{\mathbb{G}})$
```

```
PROC GETIDX$(\vec{e})$
parse $\vec{e} = (e_1, \ldots, e_w)$
for $j = 1, \ldots, w$ do
    pick first $i \in [|\mathcal{L}^E|]$
        such that $\mathcal{L}_i^E = e_j$
    $i_j := i$
return $(i_1, \ldots, i_w)$
```

```
PROC ENCODE$(G)$
parse $G = (G_1, \ldots, G_u)$
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 $(e_1, \ldots, e_u)$
```

Fig. 1. Procedures implementing the generic group oracle

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 $i \in [|\mathcal{L}^{\mathbb{G}}|]$. 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.

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, that is, if 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 $e_j = \mathcal{L}_{i_j}^E$, and returns $(i_1, \ldots, i_w)$.[4]

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

[4] Note that GETIDX may receive only encodings $e_1, \ldots, e_w$ which are already contained in $\mathcal{L}^E$, as otherwise the behavior of GETIDX is undefined. We will make sure that this is always the case.

The lists $\mathcal{L}^{\mathbb{G}}$, $\mathcal{L}^{E}$ are initially empty. Then $\mathcal{O}$ calls $(e_1, \ldots, e_u) \leftarrow_\$ \textsc{Encode}(G_1, \ldots, G_u)$ to determine encodings for all group elements $G_1, \ldots, G_u$ and starts the generic group algorithm on input $\mathcal{R}(e_1, \ldots, e_u, C')$.

$\mathcal{R}^{\mathcal{O}}$ 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 2. 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 [23]) 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 $C = (\phi(C_1), \ldots, \phi(C_\ell), C') \in \mathbb{G}^u \times \{0,1\}^*$ 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 $S' \in \{0,1\}^*$. Reductions from an *interactive* computational problem Π may additionally have access to an oracle $\mathcal{S}_\Pi$ corresponding to Π, we write $\mathcal{R}^{\mathcal{A},\mathcal{O},\mathcal{S}_\Pi}$.

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 (like [22, 6, 31], 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 is run can easily translate between the two encodings. Consider as an example the reduction shown in Figure 2 that interacts with a non-generic adversary $\mathcal{A}$. Our notion of generic reductions merely formalizes that the reduction works identically for any group representation. This is illustrated in Figure 2 with an "environment" converting group elements received and output by the reduction from one group representation to another. Note also that essentially all security reductions (from a computational problem in an algebraic group) in the literature are generic. 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.

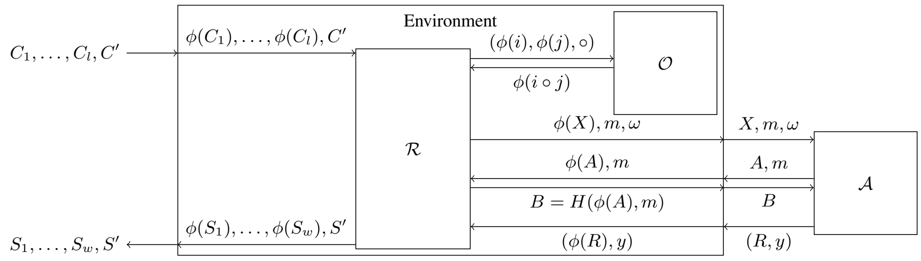

Fig. 2. An example of the interaction between a generic reduction $\mathcal{R}$ and a non-generic adversay $\mathcal{A}$ against the unforgeability of Schnorr signatures. All group elements – such as the challenge input, random oracle queries, and the signature output by $\mathcal{A}$ – are encoded by 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.

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 is run 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 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.

3 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 UF-NM-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 representation-invariant computational problem. This includes even interactive problems.

3.1 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 UF-NM 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. Later we turn to reductions that may execute $\mathcal{A}$ repeatedly, like for instance the known security proof from [22] based on the Forking Lemma.

An Inefficient Adversary $\mathcal{A}$ In this section we describe an inefficient adversary $\mathcal{A}$ that breaks the UF-NM-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 X, a message m, and random coins $\omega \in \{0,1\}^{\kappa}$.
2. The forger $\mathcal{A}$ chooses q uniformly random group elements $R_1, \dots, R_q \leftarrow_{\$} \mathbb{G}$. (We make the assumption that $q \leq |\mathbb{G}|$.) Subsequently, the forger $\mathcal{A}$ queries the random oracle $\mathcal{H}$ on (R_i, m) for all $i \in [q]$. Let $c_i := \mathcal{H}(R_i, m) \in \mathbb{Z}_p$ 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 $g^y = X^{c_\alpha} \cdot R_\alpha$, 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 X. Thus, the forger $\mathcal{A}$ breaks the UF-NM-security of the Schnorr signatures with probability 1.

Main Result for Vanilla Reductions Now we are ready to prove our main result for vanilla reductions.

Theorem 1. *Let $\Pi = (\mathcal{G}_\Pi, \mathcal{S}_\Pi, \mathcal{V}_\Pi)$ be a representation-invariant (possibly interactive) 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 an attacker $\mathcal{A}$ that $(\epsilon_{\mathcal{A}}, t_{\mathcal{A}})$-breaks the* UF-NM-*security of Schnorr signatures with success probability $\epsilon_{\mathcal{A}} = 1$ by asking q random oracle queries. Then there exists an algorithm $\mathcal{M}$ that (ϵ, t)-solves Π with $\epsilon \geq \epsilon_{\mathcal{R}} - \frac{2(u+q+t_{\mathcal{R}})^2}{p}$ and $t \approx t_{\mathcal{R}}$.*

Remark 3. Observe that Theorem 1 rules out reductions from nearly arbitrary computational problems (even interactive). At a first glance this might look contradictory, for instance there always exists a trivial reduction from the problem of forging Schnorr signatures to solving the same problem. However, as explained in Example 5, forging Schnorr-signatures is not a representation-invariant computational problem, therefore this is not a contradiction.

Proof. Assume that there exists a generic vanilla reduction $\mathcal{R} := \mathcal{R}^{\mathcal{O}, \mathcal{S}'_\Pi, \mathcal{A}}$ that $(\epsilon_{\mathcal{R}}, t_{\mathcal{R}})$-solves Π, when given access to a generic group oracle $\mathcal{O}$, an oracle $\mathcal{S}'_\Pi$, and a forger $\mathcal{A}(\phi(X), m, \omega)$, where the inputs to the forger are chosen by $\mathcal{R}$. Furthermore, the reduction $\mathcal{R}$ simulates the random oracle $\mathcal{R}.\mathcal{H}$ for $\mathcal{A}$. We show how to build a meta-reduction $\mathcal{M}$ that has black-box access to $\mathcal{R}$ and to an oracle $\mathcal{S}_\Pi$ and that 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 we 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 $\mathcal{R}$ in 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 $\mathcal{M}_0 := \mathcal{M}_0^{\mathcal{S}_\Pi}$ is an algorithm for solving a representation-invariant computational problem Π, as defined in Section 2.3. 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, has access to oracle $\mathcal{S}_\Pi$ provided by 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 $\mathcal{O}$, the $\mathcal{S}_\Pi$ oracle, 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).

```
PROC $\mathcal{M}_0(C)$
# INITIALIZATION
parse $C = (C_1, \ldots, C_u, C')$
$\mathcal{L}^{\mathbb{G}} := \emptyset$ ; $\mathcal{L}^{E} := \emptyset$
$\vec{R} = (R_1, \ldots, R_q) \leftarrow_\$ \mathbb{G}^q$
$\mathcal{I} := (C_1, \ldots, C_u, R_1, \ldots, R_q)$
ENCODE($\mathcal{I}$)
$\hat{C} := (\mathcal{L}^E_1, \ldots, \mathcal{L}^E_u, C')$
$\hat{S} \leftarrow_\$ \mathcal{R}^{\mathcal{O},\mathcal{A}}(\hat{C})$
# FINALIZATION
parse $\hat{S} := (\hat{S}_1, \ldots, \hat{S}_w, S')$
$(i_1, \ldots, i_w) :=$ GETIDX($\hat{S}_1, \ldots, \hat{S}_w$)
return $(\mathcal{L}^{\mathbb{G}}_{i_1}, \ldots, \mathcal{L}^{\mathbb{G}}_{i_w}, S')$
```

```
PROC $\mathcal{A}(\phi(X), m, \omega)$
for all $i \in [q]$
    $c_i = \mathcal{R}.\mathcal{H}(\phi(R_i), m)$
$\alpha \leftarrow_\$ [q]$
$y := \log_g X^{c_\alpha} R_\alpha$
return $(R_\alpha, y)$.

PROC $\mathcal{S}_\Pi{}'(Q)$
parse $Q = (e_1, \ldots, e_v, Q')$
$(i_1, \ldots, i_v) =$ GETIDX($e_1, \ldots, e_v$)
$(A_1, \ldots, A_\nu, A') = \mathcal{S}_\Pi(\mathcal{L}_{i_1}, \ldots, \mathcal{L}_{i_\nu}, Q')$
$(f_1, \ldots, f_\nu) =$ ENCODE($A_1, \ldots, A_\nu$)
return $(f_1, \ldots, f_\nu, A')$.
```

Fig. 3. Implementation of $\mathcal{M}_0$

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, $\mathcal{M}_0$ 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 $\mathcal{M}_0$ starts 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 2.4. 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 $e = e_i$ and $e' = e_j$ by calling $(i, j) = \text{GETIDX}(e, e')$. Then it computes $\mathcal{L}_i^{\mathbb{G}} \circ \mathcal{L}_j^{\mathbb{G}}$ and returns $\text{ENCODE}(\mathcal{L}_i^{\mathbb{G}} \circ \mathcal{L}_j^{\mathbb{G}})$.

ORACLE $\mathcal{S}'_\Pi(Q)$: This procedure handles queries issued by $\mathcal{R}$ to $\mathcal{S}'_\Pi$ by forwarding them to oracle $\mathcal{S}_\Pi$ provided by the challenger and returning the response. That is, whenever $\mathcal{R}$ submits a query $Q = (e_1, \ldots, e_v, Q') \in E^v \times \{0,1\}^*$ to $\mathcal{S}'_\Pi$, the meta-reduction runs $(i_1, \ldots, i_v) := \text{GETIDX}(e_1, \ldots, e_v)$ and queries $\mathcal{S}_\Pi$ to compute $(A_1, \ldots, A_\nu, A') := \mathcal{S}_\Pi(\mathcal{L}_{i_1}, \ldots, \mathcal{L}_{i_\nu}, Q')$. Then $\mathcal{M}_0$ determines the corresponding encodings as $(f_1, \ldots, f_\nu) := \text{ENCODE}(A_1, \ldots, A_\nu)$ and returns $(f_1, \ldots, f_\nu, A')$ to $\mathcal{R}$.

THE FORGER $\mathcal{A}(\phi(X), m, \omega)$: This procedure implements a simulation of the inefficient attacker $\mathcal{A}$ described in Section 3.1. It proceeds as follows. When $\mathcal{R}$ outputs $(\phi(X), m, \omega)$ to invoke an instance of $\mathcal{A}$, $\mathcal{A}$ queries the random oracle $\mathcal{R}.\mathcal{H}$ provided by $\mathcal{R}$ on $(\phi(R_i), m)$ for all $i \in [q]$, to determine $c_i = \mathcal{H}(\phi(R_i), m)$. Afterwards, $\mathcal{M}_0$ chooses an index $\alpha \leftarrow_\$ [q]$ uniformly at random, computes the the discrete logarithm $y := \log_g X^{c_\alpha} R_\alpha$ by exhaustive search, and outputs (R_α, y). (This step is not efficient. We show in subsequent games how to implement this attacker efficiently.)

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 $(i_1, \ldots, i_w) := \text{GETIDX}(\hat{S}_1, \ldots, \hat{S}_w)$ 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 oracles $\mathcal{O}$ and $\mathcal{S}_\Pi$ and it also mimics the attacker from Section 3.1 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 like 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 $e = e_i$ and $e' = e_j$ by calling GETIDX. It computes $a := \mathcal{L}^V_i \diamond \mathcal{L}^V_j \in \mathbb{Z}_p^{u+q}$, where $\diamond := +$ if $\circ = \cdot$ and $\diamond := -$ if $\circ = \div$, and appends a to $\mathcal{L}^V$. Finally it returns ENCODE$(\mathcal{L}^{\mathbb{G}}_i \circ \mathcal{L}^{\mathbb{G}}_j)$.

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 $\mathsf{Eval}(V, a)$ shorthand for $\mathsf{Eval}(V, a) := \prod_{i=1}^{u+q} v_i^{a_i}$ in the sequel. 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 $\mathcal{L}^{\mathbb{G}}_i = \mathsf{Eval}(\mathcal{I}, \mathcal{L}^V_i)$ for all $i \in [|\mathcal{L}^{\mathbb{G}}|]$. 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 $\mathcal{L}^{\mathbb{G}}_i := \mathsf{Eval}(\mathcal{I}, \mathcal{L}^V_i)$ for $i \in [|\mathcal{L}^V|]$. The reduction $\mathcal{R}$ is completely oblivious to this additional bookkeeping performed by $\mathcal{M}_1$, thus we have $\Pr[X_1] = \Pr[X_0]$.

```
PROC $\mathcal{M}_1(C)$
# INITIALIZATION
parse $C = (C_1, \ldots, C_u, C')$
$\mathcal{L}^{\mathbb{G}} := \emptyset$ ; $\mathcal{L}^E := \emptyset$ ; $\boxed{\mathcal{L}^V := \emptyset}$
$\vec{R} = (R_1, \ldots, R_q) \leftarrow_\$ \mathbb{G}^q$
$\mathcal{I} := (C_1, \ldots, C_u, R_1, \ldots, R_q)$
ENCODE$(\mathcal{I})$
$\boxed{\mathcal{L}^V_i := \eta_i, \ \forall i \in [u+q].}$
$\hat{C} := (\mathcal{L}^E_1, \ldots, \mathcal{L}^E_u, C')$
$\hat{S} \leftarrow_\$ \mathcal{R}^{\mathcal{O},\mathcal{A}}(\hat{C})$
# FINALIZATION
parse $\hat{S} := (\hat{S}_1, \ldots, \hat{S}_w, S')$
$(i_1, \ldots, i_w) :=$ GETIDX$(\hat{S}_1, \ldots, \hat{S}_w)$
return $(\mathcal{L}^{\mathbb{G}}_{i_1}, \ldots, \mathcal{L}^{\mathbb{G}}_{i_w}, S')$

PROC $\mathcal{O}(e, e', \circ)$
$(e, e', \circ) \in E \times E \times \{\cdot, \div\}$
$i :=$ GETIDX$(e)$
$j :=$ GETIDX$(e')$
$\boxed{a := \mathcal{L}^V_i \diamond \mathcal{L}^V_j \in \mathbb{Z}_p^{u+q}}$
$\boxed{\text{append } a \text{ to } \mathcal{L}^V}$
return ENCODE$(\mathcal{L}^{\mathbb{G}}_i \circ \mathcal{L}^{\mathbb{G}}_j)$
```

Fig. 4. Meta-Reduction $\mathcal{M}_1$. Boxed elements show the differences to $\mathcal{M}_0$. All other procedures are identical to $\mathcal{M}_0$ and thus omitted.

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 $\mathcal{A}(\phi(X), m, \omega)$: When $\mathcal{R}$ outputs $(\phi(X), m, \omega)$ to invoke an instance of $\mathcal{A}$, $\mathcal{A}$ queries the random oracle $\mathcal{R}.\mathcal{H}$ provided by $\mathcal{R}$ on $(\phi(R_i), m)$ for all $i \in [q]$, to determine $c_i = \mathcal{H}(\phi(R_i), m)$. Then it chooses an index $\alpha \leftarrow_\$ [q]$ uniformly at random, samples an element y uniformly at random from $\mathbb{Z}_p$, computes $R^*_\alpha := g^y X^{-c_\alpha}$, and re-computes the entire list $\mathcal{L}^{\mathbb{G}}$ using R^*_α instead of R_α.

More precisely, let $\mathcal{I}^* := (C_1, \dots, C_u, R_1, \dots, R_{\alpha-1}, R^*_\alpha, R_{\alpha+1}, \dots, R_q)$. 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 $i \in [|\mathcal{L}^V|]$. 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 $g^y = X^{c_\alpha} \cdot R^*_\alpha$, where $c_\alpha = \mathcal{H}(\phi(R^*_\alpha), m)$. 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.

```
PROC $\mathcal{A}(\phi(X), m, \omega)$ :
$\alpha \leftarrow_\$ [q]$
for all $i \in [q]$
    $c_i = \mathcal{R}.\mathcal{H}(\phi(R_i), m)$
$y \leftarrow_\$ \mathbb{Z}_p$ ; $R^*_\alpha := g^y X^{-c_\alpha}$
$\mathcal{I}^* := (C_1, \dots, C_u, R_1, \dots, R_{\alpha-1}, R^*_\alpha, R_{\alpha+1}, \dots, R_q)$
for $j = 1, \dots, |\mathcal{L}^{\mathbb{G}}|$ do
    $\mathcal{L}^{\mathbb{G}}_i := \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_i)$
return $(y, \phi(R^*_\alpha))$
```

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

Lemma 1. *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

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

The proof of Lemma 1 is deferred to the full version.We apply it to finish the proof of Theorem 1. By Lemma 1, the 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 $\Pr[X_2] \ge \Pr[X_1] - 2(u + q + t_{\mathcal{R}})^2/p$.

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

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

Remark 4. Note that the simulated forger *re-computes* the entire list $\mathcal{L}^{\mathbb{G}}$ after replacing R_α with R^*_α. This ensures consistency of the attacker's view before and after replacing R_α with R^*_α, *if (and only if)* it holds that

$$\mathsf{Eval}(\mathcal{I}, \mathcal{L}^V_i) = \mathsf{Eval}(\mathcal{I}, \mathcal{L}^V_j) \iff \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_i) = \mathsf{Eval}(\mathcal{I}^*, \mathcal{L}^V_j) \tag{3}$$

Lemma 1 bounds the probability that 3 does not hold, thus it bounds the probability that an attacker is able to notice the re-programming by receiving different results before and after the re-programming.

4 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 [22] is of this type.

The meta-reduction described in detail in the full version is heavily based on Seurin's meta-reduction [28]. Essentially, we show that our new simulation of forged signatures is compatible with Seurin's approach for simulating a sequence of Random Oracle queries. In combination this allows to prove that a generic reduction from *any* representation-invariant computational problem Π to breaking Schnorr signatures loses a factor of at least q, which essentially matches the upper bound of [22].

The description of the corresponding family of adversaries and the proof of the following theorem can be found in the full version.

Theorem 2. *Let Π be a representation-invariant computational problem. Suppose there exists a generic reduction $\mathcal{R}^{\mathcal{O},\mathcal{S}'_\Pi,\mathcal{A}_{F,f}}$ that $(\epsilon_{\mathcal{R}}, t_{\mathcal{R}})$-solves Π, having n-*time *black-box access to an attacker $\mathcal{A}_{F,f}$ that $(\epsilon_{\mathcal{A}}, t_{\mathcal{A}}, q)$-breaks the* UF-NM*-security of Schnorr signatures with success probability $\epsilon_{\mathcal{A}} < 1$ in time $t_{\mathcal{A}} \approx q$. Then there exists an algorithm $\mathcal{M}$ that (ϵ, t)-solves Π with $t \approx t_{\mathcal{R}}$ and*

$$\epsilon \ge \epsilon_{\mathcal{R}} - 2n(u + nq + t_{\mathcal{R}})/p - n \ln\left((1 - \epsilon_{\mathcal{A}})^{-1}\right)/q(1 - p^{-1/4})$$

This bound allows essentially the same analysis as in [28] and thus we arrive (for $\epsilon_{\mathcal{A}} \approx 1 - (1 - 1/q)^q$) at a lower bound for ϵ of approximately $\epsilon_{\mathcal{R}} - \frac{n}{q}$. Therefore, $\mathcal{R}$ must necessarily lose a factor of almost $1/q$ if the discrete logarithm problem is indeed hard.

5 A Note on Tightly-Secure Schnorr-Type Signatures

There exist several variants of Schnorr signatures with *tight* security reductions from representation-invariant computational problems. This includes, for instance, the schemes of Goh and Jarecki [16] and Chevallier-Mames [7], which are based on the computational Diffie-Hellman problem, and the scheme of Shao [29].

It is natural to ask why our tightness bound, in particular our technique of re-programming the group representation, can not be applied to these schemes. Due to space limitations, we have to defer this discussion to the full version of this paper.

Acknowledgments. We thank the anonymous reviewers for valuable comments. Nils Fleischhacker and Dominique Schröder were supported by the German Federal Ministry of Education and Research (BMBF) through funding for the Center for IT-Security, Privacy, and Accountability (CISPA; see `www.cispa-security.org`). Dominique Schröder is also supported by an Intel Early Career Faculty Honor Program Award.

References

1. Bellare, M., Namprempre, C., Pointcheval, D., Semanko, M.: The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. Journal of Cryptology 16(3), 185–215 (2003)
2. Bellare, M., Palacio, A.: GQ and schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, p. 162. Springer, Heidelberg (2002)
3. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Ashby, V. (ed.) Conference on Computer and Communications Security ACM CCS 1993, Fairfax, Virginia, USA, November 3–5, pp. 62–73. ACM Press (1993)
4. Bellare, M., Rogaway, P.: The exact security of digital signatures - how to sign with RSA and rabin. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 399–416. Springer, Heidelberg (1996)
5. Bernstein, D.J.: Proving tight security for rabin-williams signatures. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 70–87. Springer, Heidelberg (2008)
6. Boneh, D., Boyen, X.: Secure identity based encryption without random oracles. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 443–459. Springer, Heidelberg (2004)
7. Chevallier-Mames, B.: An efficient CDH-based signature scheme with a tight security reduction. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 511–526. Springer, Heidelberg (2005)
8. Coron, J.-S.: On the exact security of full domain hash. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 229–235. Springer, Heidelberg (2000)
9. Coron, J.-S.: Optimal security proofs for PSS and other signature schemes. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 272–287. Springer, Heidelberg (2002)
10. Dodis, Y., Haitner, I., Tentes, A.: On the instantiability of hash-and-sign RSA signatures. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 112–132. Springer, Heidelberg (2012)
11. Dodis, Y., Oliveira, R., Pietrzak, K.: On the generic insecurity of the full domain hash. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 449–466. Springer, Heidelberg (2005)
12. Dodis, Y., Reyzin, L.: On the power of claw-free permutations. In: Cimato, S., Galdi, C., Persiano, G. (eds.) SCN 2002. LNCS, vol. 2576, pp. 55–73. Springer, Heidelberg (2003)

13. Fischlin, M., Fleischhacker, N.: Limitations of the meta-reduction technique: The case of schnorr signatures. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 444–460. Springer, Heidelberg (2013)
14. Fischlin, M., Lehmann, A., Ristenpart, T., Shrimpton, T., Stam, M., Tessaro, S.: Random oracles with(out) programmability. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 303–320. Springer, Heidelberg (2010)
15. Garg, S., Bhaskar, R., Lokam, S.V.: Improved bounds on security reductions for discrete log based signatures. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 93–107. Springer, Heidelberg (2008)
16. Goh, E.J., Jarecki, S.: A signature scheme as secure as the Diffie-Hellman problem. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 401–415. Springer, Heidelberg (2003)
17. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal on Computing 17(2), 281–308 (1988)
18. Kakvi, S.A., Kiltz, E.: Optimal security proofs for full domain hash, revisited. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 537–553. Springer, Heidelberg (2012)
19. Maurer, U.M.: Abstract models of computation in cryptography. In: Smart, N.P. (ed.) Cryptography and Coding 2005. LNCS, vol. 3796, pp. 1–12. Springer, Heidelberg (2005)
20. Neven, G., Smart, N.P., Warinschi, B.: Hash function requirements for schnorr signatures. J. Mathematical Cryptology 3(1), 69–87 (2009)
21. Paillier, P., Vergnaud, D.: Discrete-log-based signatures may not be equivalent to discrete log. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 1–20. Springer, Heidelberg (2005)
22. Pointcheval, D., Stern, J.: Security proofs for signature schemes. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 387–398. Springer, Heidelberg (1996)
23. Reingold, O., Trevisan, L., Vadhan, S.P.: Notions of reducibility between cryptographic primitives. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 1–20. Springer, Heidelberg (2004)
24. Rupp, A., Leander, G., Bangerter, E., Dent, A.W., Sadeghi, A.-R.: Sufficient conditions for intractability over black-box groups: Generic lower bounds for generalized DL and DH problems. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 489–505. Springer, Heidelberg (2008)
25. Schäge, S.: Tight proofs for signature schemes without random oracles. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 189–206. Springer, Heidelberg (2011)
26. Schnorr, C.-P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 239–252. Springer, Heidelberg (1990)
27. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of Cryptology 4(3), 161–174 (1991)
28. Seurin, Y.: On the exact security of schnorr-type signatures in the random oracle model. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 554–571. Springer, Heidelberg (2012)
29. Shao, Z.: A provably secure short signature scheme based on discrete logarithms. Inf. Sci. 177(23), 5432–5440 (2007)
30. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 256–266. Springer, Heidelberg (1997)
31. Waters, B.: Efficient identity-based encryption without random oracles. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 114–127. Springer, Heidelberg (2005)

A Universal Unforgeability under No-Message Attacks

Consider the following security experiment involving a signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$, an attacker $\mathcal{A}$, and a challenger $\mathcal{C}$.

1. The challenger $\mathcal{C}$ computes a key-pair $(X, x) \leftarrow_{\$} \mathsf{Gen}(g)$ and chooses a message $m \leftarrow_{\$} \{0,1\}^k$ uniformly at random. It sends (X, m) to the adversary $\mathcal{A}$.
2. Eventually, $\mathcal{A}$ stops, outputting a signature σ.

Definition 4. *We say that $\mathcal{A}$ (ϵ, t)-breaks the* UF-NM*-security of* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$*, if $\mathcal{A}$ runs in time at most t and* $\Pr\left[\mathcal{A}(X, m) = \sigma : \mathsf{Vrfy}(X, m, \sigma) = 1\right] \geq \epsilon$.

Note that UF-NM-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 makes our result only stronger. In fact, if we rule out reductions from some problem Π to forging signatures in the sense of UF-NM, then the impossibility clearly holds for stronger security goals, like existential unforgeability under adaptive chosen-message attacks [17], too.