On the Multi-User Security of Short Schnorr Signatures

Jeremiah Blocki[*] and Seunghoon Lee[†]

Department of Computer Science
Purdue University
West Lafayette IN 47907, USA

Abstract. The Schnorr signature scheme is an efficient digital signature scheme with short signature lengths, i.e., $4k$-bit signatures for k-bits of security. A Schnorr signature σ over a group of size $q \approx 2^{2k}$ consists of a tuple (s, e) where $e \in \mathbb{Z}_q$ is a hash output and s must be computed using the secret key. Schnorr proposed the possibility of shorter Schnorr signatures with the same security level by truncating the hash output to k-bits, i.e., $e < 2^k$. A previous result showed that short Schnorr signatures provide k-bits of single-user security in the programmable random oracle model plus (a non-standard version of) the generic group model. Another prior result demonstrated that *standard* Schnorr signatures provide k-bits of multi-user security in the programmable random oracle model plus (another non-standard version of) the generic group model. As we discuss in the paper these non-standard versions of the generic group model do not capture all generic attacks, e.g., the generic preprocessing attacks of Corrigan-Gibbs and Kogan. In this paper we prove that short Schnorr signatures provide k-bits of (multi-user) security under the (standard) generic group model and the programmable random oracle model.

Our techniques also allow us to prove the 1-out-of-N discrete-log problem is hard even if the attacker has access to a restricted discrete-log oracle. Here the attacker is given $h_1 = g^{x_1}, \ldots, h_N = g^{x_N}$ as is challenged to output any $x \in \{x_1, \ldots, x_N\}$. The restricted oracle $\mathtt{DLog}_g(\cdot)$ will solve the discrete-log problem on any "fresh" input i.e., $\mathtt{DLog}_g(g^x) = x$ as long as $x \notin \{x_1, \ldots, x_N\}$ and g^x has not been the output of a prior query to the generic group operation. We show that any attacker running in time t wins the 1-out-of-N discrete-log problem with probability at most $\mathcal{O}\left((t+N)/2^k\right)$. Similarly, in our security reduction for (multi-user) security of short Schnorr signatures we permit the attackers to have access to the restricted discrete-log oracle $\mathtt{DLog}_g(\cdot)$.

Keywords: Schnorr Signatures · Generic Group Model · Random Oracle Model · Multi-User Security · 1-out-of-N Discrete-Log

[*] jblocki@purdue.edu
[†] lee2856@purdue.edu

1 Introduction

The Schnorr signature scheme [Sch90] has been widely used due to its simplicity, efficiency and short signature size. In the Schnorr signature scheme we start with a cyclic group G of prime order q and pick a random secret key $sk \in \mathbb{Z}_q$. To sign a message m we pick $r \in \mathbb{Z}_q$ uniformly at random, compute $I = g^r$, $e = \mathsf{H}(I||m)$ and $s = r + sk \cdot e \mod q$. The final signature is $\sigma = (s, e)$. To achieve k-bits of security we select a hash function H with $2k$-bit outputs and we select q to be a random $2k$-bit prime so that the length of a signature is $4k$-bits long. In Schnorr's original paper [Sch90] the author proposed the possibility of achieving even shorter Schnorr signatures by selecting a hash function H with k-bit outputs (or truncating to only use the first k bits) so that the final signature $\sigma = (s, e)$ can be encoded with $3k$-bits. Throughout the paper we refer to this signature scheme as the *short Schnorr signature* scheme. In this paper we investigate the following question:

> *Does the* short *Schnorr signature scheme achieve k-bits of (multi-user) security?*

Proving security for the Schnorr signatures has been a challenging task against the interactive attacks. Pointcheval and Stern [PS96] provided a reduction from the discrete-log problem in the random oracle model. However, their reduction is not tight i.e., they show that $\mathsf{Adv}_{\mathsf{sig}} \leq \mathsf{Adv}_{\mathsf{dlog}} \times q_{\mathsf{RO}}$ for any attacker making at most q_{RO} queries to the random oracle. The loss of the factor q_{RO}, which prevents us from concluding that the scheme provides k-bits of security, seems to be unavoidable e.g., see [Seu12,FJS14]. Neven et al. [NPSW09] analyzed Schnorr signatures in the generic group model [Sho97] showing that the scheme provides k-bits of security as long as the hash function satisfies two key properties random-prefix preimage (rpp) and random-prefix second-preimage (rpsp) security. Interestingly, Neven et al. [NPSW09] do not need to assume that H is a random oracle though a random oracle H would satisfy both rpp and rpsp security. Neven et al. [NPSW09] considered the short Schnorr signature scheme, but their upper bounds do not allow us to conclude that the short Schnorr signature scheme provides k-bits of security.

An earlier paper of Schnorr and Jakobsson [SJ00] analyzed the security of the short Schnorr signature scheme in the random oracle model plus (a non-standard version of) the generic group model. While they show that the scheme provides k-bits of security they also consider a restrictive version of the generic group model which is different from the original definition proposed by Shoup [Sho97]. The non-standard version they consider is restrictive and is not expressive enough to capture all known attacks, e.g., any attack that requires the ability to hash group elements including a recent preprocessing attacks of Corrigan-Gibbs and Kogan [CK18] cannot be captured in the restricted generic group model.

Galbraith et al. [GMLS02] claimed that the original Schnorr signature scheme provides tight multi-user security. In particular, they claimed to have a tight reduction showing that single-user security implies multi-user security of the original Schnorr signature scheme. However, Bernstein [Ber15] identified an error in the security proof in [GMLS02]. Bernstein [Ber15] proposed a modified "key-prefixed" version of the original Schnorr signature scheme, and proved that the "key-prefixed" version does provide multi-user security. A result of Kiltz et al. [KMP16] implies that the *original* Schnorr signature scheme provides k-bits of security in the programmable random oracle model plus (another non-standard version of) the generic group model. To the best of our knowledge the multi-user security of *short* Schnorr signatures (with or without "key-prefixing") has not been previously studied.

1.1 Our Contributions

We show that the *short* Schnorr Signature scheme provides k-bits of security against an attacker in *both* the single and multi-user versions of the signature forgery game. Our results assume the programmable random oracle model and the (standard) generic group model.

Single-User Security of Short Schnorr Signatures in GM+ROM. We first consider the single-user security of the *short* Schnorr Signature scheme. We first prove that *short* Schnorr signatures provide k-bits of security in the (original[1]) generic group model and the random oracle model.

Theorem 1 (informal). *Any attacker running in time t wins the signature forgery game* (UF-CMA) *against the short Schnorr signature scheme with probability at most $\mathcal{O}\left(t/2^k\right)$ in the generic group model (of order $q \approx 2^{2k}$) plus programmable random oracle model (See Definition 1 and Theorem 3).*

Recall that a signature scheme Π yields *k-bits of (multi-user) security* if any attacker running in time at most t can forge a signature with probability at most $\epsilon_t = t/2^k$ in the (multi-user) signature forgery game — this should hold for all time bounds $t \leq 2^k$. Theorem 1 tells us that the short Schnorr signature obtained by truncating the hash output by half would yield the same k-bits of security level with the signature length $3k$ instead of $4k$. That is, for a security level of 128 bits, the signature size would be 384 bits which is twenty-five percent less than the usual 512 bits. A 25% reduction in signature length is particularly significant in contexts where space/bandwidth is limited, e.g., on the blockchain.

Multi-User Security of Short Schnorr Signatures in GM+ROM. We show that our proofs can be extended to the multi-user case *even* in the so-called "1-out-of-N" setting, i.e., if the attacker given N public keys $pk_1, \ldots, pk_N$ can forge a signature σ which is valid under any of these public keys (it does not matter which).

Theorem 2 (informal). *Any attacker running in time t wins the multi-user signature forgery game* (MU-UF-CMA) *against the short Schnorr signature scheme with probability at most $\mathcal{O}\left((t+N)/2^k\right)$ in the generic group model (of order $q \approx 2^{2k}$) plus programmable random oracle model. Here, N denote the number of distinct users/public keys (See Definition 2 and Theorem 5).*

[1] The proof of Schnorr and Jakobsson [SJ00] gave a proof in a restricted version of the generic group model which does not capture all known attacks.

Theorem 2 guarantees that breaking multi-user security of short Schnorr signatures in "1-out-of-N" setting is not *easier* than breaking a single instance as the winning probability is still in the same order as long as $N \leq t$ which is the typical case. A naïve reduction loses a factor of N i.e., any attacker winning the multi-user forgery game with probability ϵ_{MU} can be used to win the single-user forgery game with probability $\epsilon \geq \epsilon_{\mathsf{MU}}/N$. For example, suppose that $q \approx 2^{224}$ (i.e., $k = 112$) and there are $N = 2^{32}$ instances of short Schnorr signatures which is almost the half of the entire world population. In the original single-user security game an attacker wins with probability at most $\epsilon \leq \mathcal{O}\left(t/2^k\right)$ so an an attacker running in time $t = 2^{80}$ would succeed with probability at most $\epsilon \approx 2^{-32}$. This only allows us to conclude that an attacker succeeds with probability at most $\epsilon_{\mathsf{MU}} \leq N\epsilon \approx 1$ in the multi-user security game! Our security proof implies that the attacker will succeed with probability $\epsilon_{\mathsf{MU}} \approx \epsilon \approx 2^{-32}$ in the above example. In particular, we don't lose a factor of N in the security reduction.

1.2 Our Techniques

We consider the variation of the discrete-log problem where the attacker $\mathcal{A}$ is given access to the restricted discrete-log oracle $\mathtt{DLog}$ (explained later). Our security reduction both uses the generic group model and the random oracle model. Consider the generic group model for a multiplicative cyclic group $(G = \langle g \rangle, \cdot)$ of prime order q. In the generic group model we select a random injective map $\tau : G \to \mathbb{G}$ where $\mathbb{G}$ is the set of bit strings of length ℓ (with $2^\ell \geq q$). An adversary is given the encoding $\tau(g)$ of the generator (along with the encoding $\tau(h)$ of any other public group element h) and may access to the "generic oracles" $\mathtt{Mult}(\cdot,\cdot)$ and $\mathtt{Inv}(\cdot)$. For an input $(\mathfrak{a}, \mathfrak{b}) \in \mathbb{G} \times \mathbb{G}$ the generic oracles $\mathtt{Mult}(\mathfrak{a}, \mathfrak{b})$ returns $\mathtt{Mult}(\mathfrak{a}, \mathfrak{b}) = \tau(\tau^{-1}(\mathfrak{a}) \cdot \tau^{-1}(\mathfrak{b}))$ whenever $\tau^{-1}(\mathfrak{a}), \tau^{-1}(\mathfrak{b}) \in G$. Similarly, $\mathtt{Inv}(\mathfrak{a}) = \tau(\tau^{-1}(\mathfrak{a})^{-1})$ provided that $\tau^{-1}(\mathfrak{a}) \in G$. Alternately, for group elements $a, b \in G$ the oracle $\mathtt{Mult}(\tau(a), \tau(b)) = \tau(a \cdot b)$ and $\mathtt{Inv}(\tau(a)) = \tau(a^{-1})$.

Restricted Discrete-Log Oracle in the Generic Group Model. Consider the generic group model as described above and suppose that $g, h_1 = g^{x_1}, \ldots, h_N = g^{x_N}$ are public parameters. In particular, the values $\tau(g), \tau(h_1), \ldots, \tau(h_N)$ are known to the attacker (e.g., as part of a public key) but not the corresponding discrete log solutions x_i. In our security reductions, we allow the attacker $\mathcal{A}$ to query a (restricted) discrete-log oracle $\mathtt{DLog}_g(\tau(y))$ which returns the unique value z such that $y = g^z$. However, the adversary is restricted and is only allowed to query $\mathtt{DLog}_g(\tau(y))$ *only if* $\tau(y)$ is "fresh", i.e.,

- $\tau(y)$ is not $\tau(g)$ or $\tau(h_1), \ldots, \tau(h_N)$, and
- $\tau(y)$ has not been the output of a prior random generic group query.

We remark that by restricting the discrete-log oracle to fresh queries we can rule out the trivial attack where the attacker simply queries $\mathtt{DLog}_g(\tau(h_i))$ for any $1 \leq i \leq N$. The restriction also rules out trival attacks where the attacker simply queries $\mathtt{DLog}_g(\tau(h_i g^r))$ after computing $\tau(h_i g^r)$ for some $1 \leq i \leq N$ and known value r using the $\mathtt{Mult}$ oracle.

The Known Set $\mathcal{K}$ and the Partially Known Set $\mathcal{PK}_x$. A core component of our security involves keeping track of group elements with (partially) known discrete-log solutions. For simplicity consider first a setting where the only public parameters are $\tau(g)$ and $\tau(h)$ where $h = g^x$ for an unknown parameter $x \in \mathbb{Z}_q$. We will maintain the invariant that for every group element $\mathfrak{y}$ that is the output of a random oracle query we know $a, b \in \mathbb{Z}_q$ such that $\mathfrak{y} = \tau(g^{ax+b})$ i.e., the tuple $(\mathfrak{y}, a, b)$ in recorded in a global list $\mathcal{L}$. Initially, we have $\mathcal{L} = ((\tau(g), 0, 1), (\tau(h), 1, 0))$. Intuitively, if we know $a_1, b_1, a_2, b_2 \in \mathbb{Z}_q$ such that $\mathfrak{y}_1 = \tau(g^{a_1 x + b_1})$ and $\mathfrak{y}_2 = \tau(g^{a_1 x + b_1})$ then we know that $\texttt{Mult}(\mathfrak{y}_1, \mathfrak{y}_2) = \tau(g^{(a_1+a_2)x+(b_1+b_2)})$ and can add $(\texttt{Mult}(\mathfrak{y}_1, \mathfrak{y}_2), a_1 + a_2, b_1 + b_2)$ to $\mathcal{L}$. Suppose that the attacker provides inputs $\mathfrak{y}_1$ and/or $\mathfrak{y}_2$ such that a_1, b_1 (and/or a_2, b_2) is not known. In this case $\mathfrak{y}_1$ is fresh and we may query $b_1 = \texttt{DLog}_g(\mathfrak{y}_1)$ and then add $(\mathfrak{y}_1, 0, b_1)$ to our list $\mathcal{L}$. Similarly, if $\mathfrak{y}_2$ is fresh we may query $b_2 = \texttt{DLog}_g(\mathfrak{y}_2)$ and then add $(\mathfrak{y}_2, 0, b_2)$ to $\mathcal{L}$.

We can partition $\mathcal{L}$ into two sets $\mathcal{K}$ which contains tuples of the form $(\mathfrak{y}, a = 0, b)$ and $\mathcal{PK}_x$ which contains tuples of the form $(\mathfrak{y}, a \neq 0, b)$. We define a special event BRIDGE which occurs when an element $\mathfrak{y}$ appears twice i.e., $(\mathfrak{y}, a', b') \in \mathcal{L}$ and $(\mathfrak{y}, a, b) \in \mathcal{L}$ with $(a', b') \neq (a, b)$. We can prove that the event BRIDGE occurs with negligible probability $O(t^2/q)$ at best e.g., observe that if the bridge event occurs we can solve the discrete log problem immediately i.e., $a'x + b' = ax + b \Rightarrow x = (b' - b)(a - a')^{-1}$ where the inverse is taken over $\mathbb{Z}_q$. Otherwise, x can (essentially) be viewed as a uniformly random element from $\mathbb{Z}_q$ subject to a few restrictions, i.e., for any pair of non-colliding records $(\mathfrak{a}_1, a_1, b_1) \in \mathcal{L}$ and $(\mathfrak{a}_2, a_2, b_2) \in \mathcal{L}$ we know that $x \neq (b_2 - b_1)(a_1 - a_2)^{-1}$.

Our approach extends to the multi-user case. In particular, given public parameters $\tau(h_1), \ldots, \tau(h_N)$ such that $h_i = g^{x_i}$ (x_i unknown) we will maintain the invariant that for every output $\mathfrak{y}$ of a generic group oracle the list $\mathcal{L}$ includes a tuple $(\mathfrak{y}, \boldsymbol{a}, b)$ such that $\mathfrak{y} = \tau(g^{\boldsymbol{a} \cdot \boldsymbol{x} + b})$ where $\boldsymbol{a}$ and $\boldsymbol{x} = (x_1, \ldots, x_N)$ are N-dimensional vectors. We can then define the known set $\mathcal{K}^N$ and the partially known set $\mathcal{PK}^N_{\{x_i\}_{i=1}^N}$ in the same way e.g., if $\boldsymbol{a} = \boldsymbol{0}$ then the tuple $(\mathfrak{y}, \boldsymbol{a}, b) \in \mathcal{K}^N$ otherwise the tuple is in $\mathcal{PK}^N_{\{x_i\}_{i=1}^N}$. We can also define the event BRIDGE^N which occurs when an element $\mathfrak{y}$ appears twice i.e., $(\mathfrak{y}, \boldsymbol{a}', b') \in \mathcal{L}$ and $(\mathfrak{y}, \boldsymbol{a}, \boldsymbol{b}) \in \mathcal{L}$ with $(\boldsymbol{a}', b') \neq (\boldsymbol{a}, b)$. We can then argue that the event bridge occurs with probability at most $O((t + N)^2/q)$. If the event BRIDGE^N does not occur then we can view $\boldsymbol{x} \in \mathbb{Z}_q^N$ as a uniformly random vector subject to a few restrictions i.e., for any pair of non-colliding records $(\mathfrak{a}_1, \boldsymbol{a}_1, b_1) \in \mathcal{L}$ and $(\mathfrak{a}_2, \boldsymbol{a}_2, b_2) \in \mathcal{L}$ we know that $\boldsymbol{x} \cdot (\boldsymbol{a}_1 - \boldsymbol{a}_2) \neq (b_2 - b_1)$. See Section 4 for more discussion and analysis.

1.3 Related Work

Security Proofs in the Random Oracle Model. Pointcheval and Stern [PS96] provided the security proofs for signature schemes in the random oracle model [BR93] using the forking lemma under the assumption that the discrete-log problem is hard. However, the security reduction is not tight in their proof. In [Seu12], it has been shown that the non-tight reduction cannot be improved under a certain assumption called the One-More Discrete Logarithm (OMDL) assumption. Going one step further, [FJS14] showed that no "generic" reduction for Schnorr signatures can be tight.

Security Proofs in the Generic Group Model. The generic group model goes back to Nechaev [Nec94] and Shoup [Sho97]. Nechaev [Nec94] proved that the discrete-log problem is hard in the generic group model. One motivation for analyzing cryptographic protocols in the generic group model is that for certain elliptic curve groups the best known attacks are all generic [JMV01,FST10]. Dent [Den02] showed that there are protocols which are provably secure in the generic group model but which are trivially insecure when the generic group is replaced with any (efficiently computable) real one. However, these results were artificially crafted to provide a counter-example. Similar to the random oracle model [BR93] experience suggests that protocols with security proofs in the generic group model do no have inherent structural weaknesses and will be secure as long as we instantiate with a reasonable elliptic curve group. See [KM07,Fis00,JS08] for additional discussion of the strengths/weaknesses of proofs in the generic group model.

In [NPSW09] the security proof of the Schnorr signatures in the generic group model has been provided assuming that the hash function $\mathsf{H}(\cdot)$ satisfies certain properties such as random-prefix preimage (rpp) security and random-prefix second-preimage (rpsp) security instead of being a random oracle. The authors mention the possibility of short Schnorr signatures by truncating the hash output bits by k, however, the security reduction is not tight and there exists a tradeoff between the hash output bits k and the group size q as mentioned before.

Recently, Corrigan-Gibbs and Kogan [CK18] used the generic group model to analyze the security of several cryptographic problems (e.g., discrete-log, computational/decisional Diffie-Hellman, etc...) against preprocessing attacks. We do not focus on the security of (short) Schnorr signatures against preprocessing attacks though this could be an interesting direction for future work. Our results do imply that any attacker, given N public signing keys, cannot forge any signature except with probability $\mathcal{O}\left((t+N)/2^k\right)$. We remark that an attacker running in time $\tilde{\mathcal{O}}\left(2^k\sqrt{N}\right)$ can extract all N secret keys, but we do not know if this is tight, i.e., we do not rule out the possibility that an attacker running in time $t = 2^k$ can successfully extract all N secret keys.

Deployment of Schnorr Signatures. The Transport Layer Security (TLS) standard for the secure HTTPS connections has been maintained by the Internet Engineering Task Force (IETF), and IETF decided that the standard for the Schnorr signatures should require that the public key be included inside the hash function. Later on, Kiltz et al. [KMP16] gave a tight security reduction without including the public key in the hash function. However, the security reduction is in the random oracle model and loses a factor of roughly Q_h, the number of hash queries. As they made a stronger assumption, it is not yet clear whether IETF will drop the public key from the hash input in the TLS standard [CKMS17].

Derler and Slamanig [DS19] initiated the study of key-homomorphic signature schemes and showed a tight reduction from single-user security to "key-prefixed" multi-user security for a class of schemes including Schnorr, admitting a certain key-homomorphism. In this paper we focus on the original version of Schnorr signatures, i.e., without "key-prefixing."

Other Short Signatures. Boneh et al. [BLS04] proposed even shorter signatures called BLS signatures which is as shorter as $2k$ bits to yield k-bits of security which is secure against existential forgery under a chosen-message attack in the random oracle model assuming that the Computational Diffie-Hellman (CDH) problem is hard on certain elliptic curves over a finite field. While BLS signatures yield even shorter signature length than Schnorr signatures, the computation costs for the BLS verification algorithm is several orders of magnitude higher due to the reliance on bilinear pairings. If we allow for "heavy" cryptographic solutions such as indistinguishability obfuscation [GGH+13] (practically infeasible at the moment) then it becomes possible to achieve k-bit signatures with k-bits of security [SW14,RW14,LM17].

Generic/Non-Generic Discrete-Log Algorithms. It is well known that in Shoup's standard generic group model [Sho97] the discrete-log problem in a group of prime order q needs time $\Omega(\sqrt{q})$ to solve. For certain elliptic curve groups, these lower bounds are the best known [WZ11,BL12,GWZ15]. A number of popular discrete-log algorithms can be captured by Shoup's model, e.g., Baby-Step Giant-Step algorithm by Shanks [Sha71], Pollard's Rho and Kangaroo algorithms [Pol78], and the Pohlig-Hellman algorithm [PH06]. However, not all discrete-log algorithms are capturable by Shoup's model. In certain groups (e.g., subgroups of $\mathbb{Z}_q^*$) there exist non-generic algorithms for the discrete-log problem which outperforms the generic ones, e.g., see [GHS02,MVO91,Sma99].

1.4 Organization of This Paper

In Section 2, we recall the generic group model and the Schnorr signature scheme, define the short Schnorr signatures and the known and the partially known set for the security analysis. In Section 3, we give the security definition and the single-user security proof of short Schnorr signatures, and in Section 4, we prove the multi-user security of short Schnorr signatures. In Section 5, we analyze the security of other Fiat-Shamir-based signatures, and in Section 6, we introduce some (non-standard) generic group models and a generic attack which is not able to be captured by these models.

2 Preliminaries

2.1 The Generic Group Model

The generic group model is an idealized cryptographic model proposed by Shoup [Sho97]. Let $G = \langle g \rangle$ be a cyclic group of prime order q where g is a generator of the group. Here, we can assume that G is multiplicative. Each element of G will be encoded by bit strings of length ℓ in a cryptographic scheme. Let $\mathbb{G}$ be a set of bit strings of length ℓ, then the map $\tau : G \to \mathbb{G}$ gives the natural representation of G in $\mathbb{G}$. The key idea here is that the map τ does not need to be a group homomorphism because any adversary who attacks a primitive is only available to see a randomly chosen encoding of each group element. Hence, the generic group model assumes that an adversary has no access to the concrete representation of the group elements. Instead, the adversary is given access to an oracle parametrized by τ which computes the group operation indirectly in G as well as the encoding of the generator g given as $\mathfrak{g} = \tau(g)$. More precisely, for an input $(\mathfrak{a}, \mathfrak{b}) \in \mathbb{G} \times \mathbb{G}$, the oracles $\texttt{Mult}(\mathfrak{a}, \mathfrak{b}), \texttt{Inv}(\mathfrak{a})$, act as following:

$$\begin{aligned} \texttt{Mult}(\mathfrak{a}, \mathfrak{b}) &= \tau(\tau^{-1}(\mathfrak{a}) \cdot \tau^{-1}(\mathfrak{b})), \\ \texttt{Inv}(\mathfrak{a}) &= \tau(\tau^{-1}(\mathfrak{a})^{-1}), \end{aligned}$$

if $\tau^{-1}(\mathfrak{a}), \tau^{-1}(\mathfrak{b}) \in G$. Remark that the adversary has no access to the map τ itself and does not know what is going on in a group G. Hence, the adversary has no sense that which element in G maps to $\texttt{Mult}(\mathfrak{a}, \mathfrak{b})$ in $\mathbb{G}$ even if she sees the oracle output.

For convenience we will use the notation $\texttt{Pow}(\mathfrak{a}, k) = \tau(\tau^{-1}(\mathfrak{a})^k)$. Without loss of generality, we do not allow the attacker to directly query `Pow` as an oracle since the attacker can efficiently evaluate this subroutine using the `Mult` oracle. In particular, one can evaluate $\texttt{Pow}(\mathfrak{a}, k)$ using just $\mathcal{O}(\log k)$ calls to the multiplication oracle `Mult` using the standard modular exponentiation algorithm.

2.2 The Schnorr Signature Scheme

The Schnorr signature scheme is a digital signature scheme consists of a tuple of PPT algorithms $\Pi = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vfy})$, where $\mathsf{Kg}(1^k)$ is a key-generation algorithm to generate a public key pk and a secret key sk for security parameter k, $\mathsf{Sign}(sk, m)$ is a signing algorithm which generates a signature σ on a message $m \in \{0,1\}^*$, and $\mathsf{Vfy}(pk, m, \sigma)$ is a verification algorithm which outputs 1 if the signature is valid and 0 otherwise.

Throughout this paper, we will consider the notion of the generic group model in the Schnorr signature scheme. The corresponding scheme is described in Figure 1.

We remark that verification works for a correct signature $\sigma = (s, e)$ because $\texttt{Mult}(\texttt{Pow}(\mathfrak{g}, s), \texttt{Pow}(\texttt{Inv}(pk), e)) = \tau\left(g^{s - sk \cdot e}\right) = I$.

Short Schnorr Signatures. Typically, it is assumed that the random oracle $\mathsf{H}(I \| m)$ outputs a uniformly random element $e \in \mathbb{Z}_q$ where q is a random $2k$-bit prime. Thus, we would need $2k$-bits to encode e. To produce a shorter signature we can assume that $\mathsf{H}(I \| m)$ outputs a uniformly random integer $e \in \mathbb{Z}_{2^k}$ which can be described with just k-bits. In practice the shorter random oracle is *easier* to implement in practice since we do not need to worry about rounding issues when converting a binary string to $\mathbb{Z}_{2^k}$ i.e., we can simply take the first k bits

```
$\mathsf{Kg}(1^k)$:
  1: $sk \leftarrow_{\$} \mathbb{Z}_q$   // $q$ is $2k$-bit prime
  2: $pk \leftarrow \mathtt{Pow}(\mathfrak{g}, sk)$
  3: return $(pk, sk)$

$\mathsf{Sign}(sk, m)$:
  1: $r \leftarrow_{\$} \mathbb{Z}_q;\ I \leftarrow \mathtt{Pow}(\mathfrak{g}, r)$
  2: $e \leftarrow \mathsf{H}(I||m)$
  3: $s \leftarrow r + sk \cdot e \mod q$
  4: return $\sigma = (s, e)$

$\mathsf{Vfy}(pk, m, \sigma)$:
  1: $R \leftarrow \mathtt{Mult}(\mathtt{Pow}(\mathfrak{g}, s), \mathtt{Pow}(\mathtt{Inv}(pk), e))$   // $= \tau(g^s \cdot g^{-sk \cdot e})$
  2: if $\mathsf{H}(R||m) = e$ then
  3:     return 1
  4: else return 0
```

Figure 1. The Schnorr Signature Scheme in the Generic Group Model. Remark that the procedures by calling multiple generic oracles are highlighted in red.

of our random binary string. The result is a signature $\sigma = (s, e)$ which can be encoded in $3k$-bits — $2k$-bits to encode $s \in \mathbb{Z}_q$ plus k bits to encode e. This natural modification is straightforward and is not new to our paper. The key question we investigate is whether or not short schnorr signatures can provide k-bits of security.

3 Single-User Security of Short Schnorr Signatures

As a warm-up to our main result we first prove that short Schnorr Signatures of length $3k$ achieve k-bits of security. We first describe the standard signature forgery experiment $\mathsf{SigForge}^{\mathtt{GO}}_{\mathcal{A},\Pi}(k)$ in the generic group model and the random oracle model. Here, an attacker is given the public key $pk = \tau(g^{sk})$ along with $\tau(g)$ (the encoding of) the group generator g. The attacker is given oracle access to the signing oracle $\mathsf{Sign}(\cdot)$ as well as the generic group oracles $\mathtt{GO} = (\mathtt{Mult}(\cdot,\cdot), \mathtt{Inv}(\cdot))$ and the random oracle $\mathsf{H}(\cdot)$. The attacker's goal is to eventually output a forgery $(m, \sigma = (s, e))$ for a *fresh* message m that has not previously been submitted to the signing oracle.

Generic Signature Forgery Game. Recall that a generic group $\mathbb{G}$ is defined as a set of bit strings of length ℓ and $\tau : G \to \mathbb{G}$ be the map from a cyclic group G of prime order q to $\mathbb{G}$. Consider the following experiment defined for a signature scheme $\Pi = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vfy})$, the generic oracles $\mathtt{GO} = (\mathtt{Mult}(\cdot,\cdot), \mathtt{Inv}(\cdot))$, and adversary $\mathcal{A}$:

The Generic Signature Forgery Game $\mathsf{SigForge}^{\mathtt{GO}}_{\mathcal{A},\Pi}(k)$:

(1) $\mathsf{Kg}(1^k)$ is run to obtain the public and the secret keys (pk, sk). Here, sk is chosen randomly from the group $G = \mathbb{Z}_q$ where q is a $2k$-bit prime, and $pk = \tau(g^{sk})$ where g is the generator of the group G.

(2) Adversary $\mathcal{A}$ is given $(\mathfrak{g} = \tau(g), pk, q)$ and access to the generic group oracles $\mathtt{GO} = (\mathtt{Mult}(\cdot,\cdot), \mathtt{Inv}(\cdot))$, the random oracle $\mathsf{H}(\cdot)$ and the signing oracle $\mathsf{Sign}(\cdot)$. After multiple access to these oracles, the adversary outputs $(m, \sigma = (s, e))$.

(3) $\mathcal{A}$ succeeds to forge a signature if and only if $\mathsf{Vfy}(pk, m, \sigma) = 1$ and the message m was not previously queried by $\mathcal{A}$. We define that $\mathsf{SigForge}^{\mathtt{GO}}_{\mathcal{A},\Pi}(k) = 1$ when $\mathcal{A}$ succeeds.

We remark that a signature scheme Π yields *k-bits of security* if any attacker running in time at most t can forge a signature with probability at most $\epsilon_t = t/2^k$ and this should hold for all $t \le 2^k$. Definition 1 formalizes this argument in the sense that an attacker forges a signature if and only if $\mathsf{SigForge}^{\mathtt{GO}}_{\mathcal{A},\Pi}(k) = 1$.

Definition 1. *We say that a signature scheme $\Pi = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vfy})$ is* $(t, q_{\mathtt{RO}}, q_{\mathtt{GO}}, q_{\mathsf{Sign}}, \epsilon)$-UF-CMA secure (unforgeable against chosen message attack) *if for every adversary $\mathcal{A}$ running in time at most t and making at most $q_{\mathtt{RO}}$ (resp. $q_{\mathtt{GO}}$, q_{Sign}) queries to the random oracle (resp. generic group, signature oracles), the following bound holds:*

$$\Pr\left[\mathsf{SigForge}^{\mathtt{GO}}_{\mathcal{A},\Pi}(k) = 1\right] \le \epsilon.$$

3.1 Discrete Log Problem with Restricted Discrete Log Oracle

We prove security of short Schnorr signatures by reductions from a special variant of the discrete-log problem. It is well known that the regular discrete-log problem (given $\tau(g^x)$ find x) is hard in the generic group model. We consider a simple variant of the problem where the attacker is given access to a (restricted) discrete-log oracle $\mathtt{DLog}_g(\tau(y))$ which will output x' s.t. $g^{x'} = y$ as long as $\tau(y)$ is a "fresh" group element, i.e., $\tau(y) \ne \tau(g^x)$ and $\tau(y)$ has not previously been the output of a previous generic group query. We prove that the discrete-log problem remains hard even if we give the attacker access to this restricted oracle. The formal definition of the discrete log experiment $\mathsf{DLogChal}^{\mathtt{GO}}_{\mathcal{A}}(k)$ is given below:

The Generic Discrete-Log Game $\mathsf{DLogChal}^{\mathtt{GO}}_{\mathcal{A}}(k)$:

(1) The adversary $\mathcal{A}$ is given $\mathfrak{g} = \tau(g)$, $\tau(h) = \tau(g^x)$ for a random value of x. Here, g is a generator of a cyclic group $G = \langle g \rangle$ of order $q \approx 2^{2k}$ and $\tau : G \to \mathbb{G}$ is the map from G to a generic group $\mathbb{G}$.

(2) $\mathcal{A}$ is allowed to query the usual generic group oracles $(\mathtt{Mult}, \mathtt{Inv})$ and is additionally allowed to query $\mathtt{DLog}_g(\tau(y))$, but *only* if $\tau(y)$ is "fresh", i.e., $\tau(y)$ is not $\tau(g)$ or $\tau(h)$ and $\tau(y)$ has not been the output of a previous random generic group query.

(3) After multiple queries, $\mathcal{A}$ outputs x'.

(4) The output of the game is defined to be 1 if $x' = x$, and 0 otherwise. We write $\mathsf{DLogChal}^{\mathtt{GO}}_{\mathcal{A}}(k) = 1$ if the output of the game is 1 and in this case we say that $\mathcal{A}$ *succeeds.*

Lemma 1 upper bounds the probability that an attacker wins the generic discrete-log game $\mathsf{DLogChal}_{\mathcal{A}}^{\mathtt{GO}}(k)$.

Lemma 1. *The probability the attacker running in time t wins the generic discrete-log game (even with access to the restricted* $\mathtt{DLog}$ *oracle) is at most*

$$\Pr\left[\mathsf{DLogChal}_{\mathcal{A}}^{\mathtt{GO}}(k) = 1\right] \leq \frac{3(t+1)t+3}{q-(3t+2)^2}$$

where q is the order of the group G.

Proof Sketch: Intuitively, the proof works by modifying the original attacker $\mathcal{A}$ to maintain the invariant that for every output $\tau(y)$ that of a generic group query we can express the discrete-log of y as $y = g^{ax+b}$ for *known* constants a and b. If we know that $y_1 = g^{a_1x+b_1}$ and $y_2 = g^{a_2x+b_2}$ then we know that the output $\mathtt{Mult}(\tau(y_1), \tau(y_2)) = \tau(g^{(a_1+a_2)x+(b_1+b_2)})$. If we don't know a_1, b_1 such that $y_1 = g^{a_1x+b_1}$ then $\tau(y)$ must be "fresh" in which case we will simply set $a_1 = 0$ and query for $b_1 = \mathtt{DLog}_g(\tau(y))$ before forwarding the query $\mathtt{Mult}(\tau(y_1), \tau(y_2))$ to the generic group oracle. In our security analysis we keep track of $\mathcal{K}$ which consists of tuples $(\tau(y), a, b)$ with $a = 0$ (elements for which we *know* the discrete-log solution) and $\mathcal{PK}_x$ which consists of tuples of the form $(\tau(y), a, b)$ with $a \neq 0$ (elements for which the discrete-log solution is partially known). The key-component of the proof is to introduce an event BRIDGE which is the event that elements in $\mathcal{K}$ and $\mathcal{PK}$ collide i.e., either $\mathtt{Mult}(\tau(y_1), \tau(y_2)) \in \mathcal{PK}_x$ while $\tau(y_1), \tau(y_2) \in \mathcal{K}$ (i.e., we already know b_1, b_2 s.t. $y_1y_2 = g^{b_1+b_2}$) or $\mathtt{Mult}(\tau(y_1), \tau(y_2)) \in \mathcal{K}$ whenever we already know a_1, a_2, b_1, b_2 with $a_1 + a_2 \neq 0$ such that $y_1y_2 = g^{(a_1+a_2)x+(b_1+b_2)}$. If the event BRIDGE occurs we can solve the discrete-log challenge. Conditioning on the event $\overline{\mathsf{BRIDGE}}$ we can show that any attacker succeeds with negligibly small probability at most $\frac{1}{q-(3t+2)^2}$. We then show that the probability of the event BRIDGE itself is at most $\frac{3(t+1)t+2}{q-(3t+2)^2}$. The full proof of Lemma 1 is in Appendix B.

3.2 Security Reduction

Given Lemma 1 we are now ready to describe our security reduction for short Schnorr signatures. As in our security proof for the discrete-log problem we will ensure that for every output $\tau(y)$ of a generic group query we can express $y = g^{ax+b}$ for known constants a and b — here x is the secret key that is selected in the security game i.e., any time $\mathcal{A}_{\mathsf{sig}}$ makes a query involving a fresh element $\tau(y)$ we will simply query $\mathtt{DLog}_g(\tau(y))$ so that we can add $\tau(y)$ to our known set $\mathcal{K} \subset \mathcal{L}$.

Theorem 3 provides the first rigorous proof of the folklore claim that short $3k$-bit Schnorr signatures can provide k-bits of security. The formal security proof uses *both* the generic group model and the random oracle model:

Theorem 3. *In the generic group model of prime order $q \approx 2^{2k}$ and the programmable random oracle model the short Schnorr signature scheme is* $(t, q_{\mathtt{RO}}, q_{\mathtt{GO}}, q_{\mathsf{Sign}}, \epsilon)$-$\mathsf{UF\text{-}CMA}$ *secure with* $\epsilon = \frac{3t^2+4t+3}{q-(3t+2)^2} + \frac{t^2}{q} + \frac{t+1}{2^k} = \mathcal{O}\left(\frac{t}{2^k}\right)$.

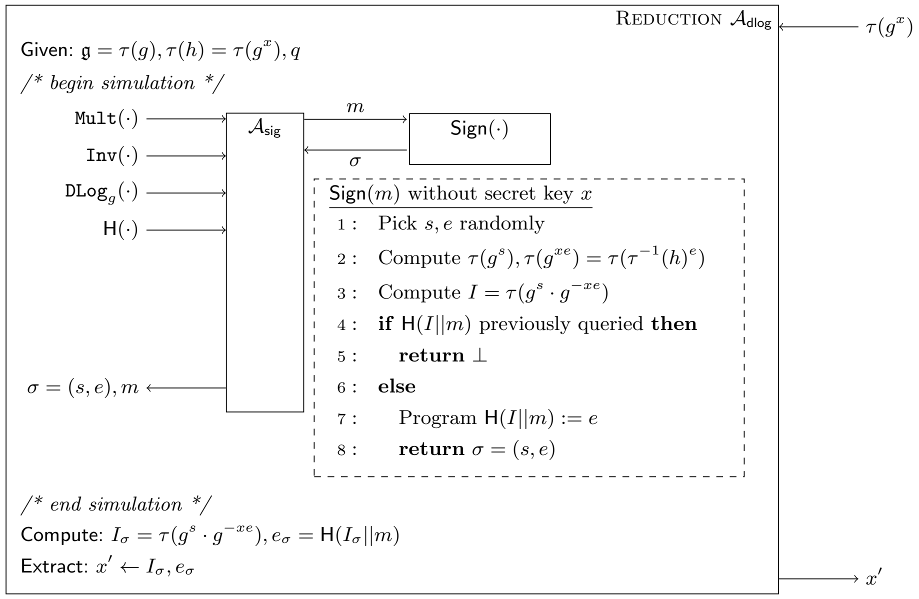
Proof. Let $\Pi = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vfy})$ be the Schnorr signature scheme. Given an adversary $\mathcal{A}_{\mathsf{sig}}$ attacking Schnorr signature scheme, we construct the following efficient algorithm $\mathcal{A}_{\mathsf{dlog}}$ which solves the discrete-logarithm problem relative to the generic group $\mathbb{G}$:

Algorithm $\mathcal{A}_{\mathsf{dlog}}$:
The algorithm is given $\mathfrak{g} = \tau(g), \tau(h) = \tau(g^x), q$ as input.

1. Initialize the list $\mathcal{L} = \{(\tau(h), 1, 0), (\tau(g), 0, 1)\}$ and $H_{\mathsf{resp}} = \{\}$ where H_{resp} stores the random oracle queries.
2. Run $\mathcal{A}_{\mathsf{sig}}$ with a number of access to the generic oracles $\mathtt{GO} = (\mathtt{Mult}(\cdot,\cdot), \mathtt{Inv}(\cdot))$, $\mathsf{Sign}(\cdot)$, and $\mathsf{H}(\cdot)$. The signing oracle without a secret key is described in Figure 2. Now we consider the following cases:
   (a) Whenever $\mathcal{A}_{\mathsf{sig}}$ submits a query x to the random oracle H:
   – If there is a pair $(x, r) \in H_{\mathsf{resp}}$ for some string r then return r.
   – Otherwise, select $r \in \mathbb{Z}_{2^k}$ uniformly at random and add (x, r) to the set H_{resp}.
   – If x has the form $x = (\mathfrak{a}||m)$ where the value $\mathfrak{a}$ has not been observed previously (i.e., is not in the list $\mathcal{L}$) then we query $b = \mathtt{DLog}_g(\mathfrak{a})$ and add $\left(\tau(g^b), 0, b\right)$ to $\mathcal{L}$.
   (b) Whenever $\mathcal{A}_{\mathsf{sig}}$ submits a query $\mathfrak{a}$ to the generic group oracle $\mathtt{Inv}(\mathfrak{a})$:
   – First check if $\mathfrak{a}$ is not in the list $\mathcal{L}$. If so we immediately query $b = \mathtt{DLog}_g(\mathfrak{a})$ and add $\left(\tau(g^b), 0, b\right)$ to $\mathcal{L}$.
   – Otherwise $(\mathfrak{a}, a, b) \in \mathcal{L}$ (i.e., $\mathfrak{a} = \tau(g^{ax+b})$). In this case we query $\mathtt{Inv}(\mathfrak{a}) = \tau(g^{-ax-b})$, output the result and add the result $\left(\tau(g^{-ax-b}), -a, -b\right) \in \mathcal{L}$.
   (c) Whenever $\mathcal{A}_{\mathsf{sig}}$ submits a query $\mathfrak{a}, \mathfrak{b}$ to the generic group oracle $\mathtt{Mult}(\mathfrak{a}, \mathfrak{b})$:
   – First, if the element $\mathfrak{a}$ (resp. $\mathfrak{b}$) is not in the list $\mathcal{L}$ then query $b_0 = \mathtt{DLog}_g(\mathfrak{a})$ (resp. $b_1 = \mathtt{DLog}_g(\mathfrak{b})$) and add the element $(\mathfrak{a}, 0, b_0)$ (resp. $(\mathfrak{b}, 0, b_1)$) to $\mathcal{L}$.
   – Otherwise both elements $(\mathfrak{a}, a_0, b_0), (\mathfrak{b}, a_1, b_1) \in \mathcal{L}$. Then we return $\mathtt{Mult}(\mathfrak{a}, \mathfrak{b}) = \tau(g^{(a_0+a_1)x+b_0+b_1})$ and add the element $(\tau(g^{(a_0+a_1)x+b_0+b_1}), a_0 + a_1, b_0 + b_1) \in \mathcal{L}$.
   (d) Whenever $\mathcal{A}_{\mathsf{sig}}$ submits a query m to the signing oracle $\mathsf{Sign}(\cdot)$:
   – We use the procedure described in Figure 2 to forge a signature without knowledge of the secret key x. Intuitively, the forgery procedure relies on our ability to program the random oracle.
   – We remark that a side effect of querying the Sign oracle is the addition of the triples $(\tau(g^s), 0, s)$, $(\tau(g^{xe}), e, 0)$ and $(\tau(g^{s-xe}), -e, s)$ to $\mathcal{L}$ since these values are computed using the generic group oracles $\mathtt{Inv}, \mathtt{Mult}$.
   (e) If at any point we have some string $\mathfrak{y}$ such that $(\mathfrak{y}, a, b) \in \mathcal{L}$ and $(\mathfrak{y}, c, d) \in \mathcal{L}$ for $(a, b) \neq (c, d)$ then we can immediately return $x = (d-b)(a-c)^{-1}$.[2] Thus, without loss of generality, we can assume that each string $\mathfrak{y}$ occurs at most once in the list $\mathcal{L}$.

[2] Note that $(a, b) \neq (c, d)$ implies $a \neq c$ since if $a = c$ then $g^{ax+b} = g^{ax+d}$ implies $b = d$ as $a, b, c, d \in \mathbb{Z}_q$.

3. After $\mathcal{A}_{\mathsf{sig}}$ outputs $\sigma = (s, e)$ and m we first compute $I_\sigma = \texttt{Mult}(\texttt{Pow}(\tau(g), s), \texttt{Inv}(\texttt{Pow}(h, e))) = \tau(g^s \cdot g^{-xe})$ and then check to see if we previously had any triple of the form $(I_\sigma, a, b) \in \mathcal{L}$.
   (a) If no such triple exists we return $\perp$.
   (b) Otherwise, we let a, b be given such that $I_\sigma = \tau(g^{ax+b})$.
      – If $a + e = 0$ then we return $\perp$.
      – Otherwise, we return $(s - b)(a + e)^{-1}$.



Reduction $\mathcal{A}_{\mathsf{dlog}}$
$\tau(g^x)$
Given: $\mathfrak{g} = \tau(g), \tau(h) = \tau(g^x), q$
/* begin simulation */
$\texttt{Mult}(\cdot)$
$\texttt{Inv}(\cdot)$
$\texttt{DLog}_g(\cdot)$
$\mathsf{H}(\cdot)$
$\mathcal{A}_{\mathsf{sig}}$
m
σ
$\mathsf{Sign}(\cdot)$
$\sigma = (s, e), m$
/* end simulation */
Compute: $I_\sigma = \tau(g^s \cdot g^{-xe}), e_\sigma = \mathsf{H}(I_\sigma || m)$
Extract: $x' \leftarrow I_\sigma, e_\sigma$
x'

```
Sign(m) without secret key x
Pick $s, e$ randomly
Compute $\tau(g^s), \tau(g^{xe}) = \tau(\tau^{-1}(h)^e)$
Compute $I = \tau(g^s \cdot g^{-xe})$
if $\mathsf{H}(I||m)$ previously queried then
    return $\perp$
else
    Program $\mathsf{H}(I||m) := e$
    return $\sigma = (s, e)$
```

Figure 2. A reduction to the discrete-log attacker $\mathcal{A}_{\mathsf{dlog}}$ from the Schnorr signature attacker $\mathcal{A}_{\mathsf{sig}}$.

Analysis. We first remark that if the signature is valid then we must have $e = \mathsf{H}(I_\sigma||m)$ and $\texttt{DLog}_g(I_\sigma) = s - xe = ax + b$ or equivalently $s - b = (a + e)x$. Thus, as long as $(a + e)$ is invertible we will have $x = (s - b)(a + e)^{-1}$.

Now consider the upper bound of the probability that our algorithm outputs $\perp$ for failure before $\mathcal{A}_{sig}$ outputs a signature as well as the probability our algorithm outputs $\perp$ after $\mathcal{A}_{sig}$ outputs a valid signature.

(1) One way to output failure is during the signing oracle if $\mathsf{H}(I||m)$ has been queried previously (Algorithm 2.(d)). We define this event as $\mathsf{FailtoSign}$. Then we have the following claim to upper bound the probability of the event:

Claim 1. $\Pr[\mathsf{FailtoSign}] \leq q_{\mathsf{Sign}} \times \dfrac{q_{\texttt{RO}} + q_{\mathsf{Sign}}}{q}$.

(2) The second way to output failure is if the value I_σ does not previously appear in either set $\mathcal{PK}_x$ or $\mathcal{K}$ (Algorithm 3.(a)). We use $\mathsf{FailtoFind}^N(I_\sigma)$ to denote the event that the signature is valid but the value I_σ does not appear in $\mathcal{L}$. Then we have:

Claim 2. $\Pr[\mathsf{FailtoFind}(I_\sigma)] \le \frac{q_{\mathtt{RO}} + q_{\mathsf{Sign}}}{q - |\mathcal{L}|} + \frac{1}{2^k}$.

(3) Finally, we could output failure if $a = -e$ (Algorithm 3.(b)). We call this event $\mathsf{BadQuery}$. Then we have:

Claim 3. $\Pr[\mathsf{BadQuery}] \le \frac{q_{\mathtt{RO}}}{2^k}$.

We defer the proofs of Claim 1, Claim 2, and Claim 3 to Appendix B. Now we have shown that

$$\begin{aligned}\Pr[\mathsf{DLogChal}^{\mathtt{GO}}_{\mathcal{A}_{\mathsf{dlog}}}(k) = 1] \ge \Pr[\mathsf{SigForge}^{\mathtt{GO}}_{\mathcal{A}_{\mathsf{sig}},\Pi}(k) = 1] - \Pr[\mathsf{FailtoSign}]&\\ - \Pr[\mathsf{FailtoFind}(I_\sigma)] - \Pr[\mathsf{BadQuery}]&\\ \ge \Pr[\mathsf{SigForge}^{\mathtt{GO}}_{\mathcal{A}_{\mathsf{sig}},\Pi}(k) = 1] - \frac{q_{\mathsf{Sign}}(q_{\mathtt{RO}} + q_{\mathsf{Sign}})}{q}&\\ - \frac{q_{\mathtt{RO}} + q_{\mathsf{Sign}}}{q - |\mathcal{L}|} - \frac{1}{2^k} - \frac{q_{\mathtt{RO}}}{2^k}&\ .\end{aligned}$$

Since $q_{\mathtt{RO}} + q_{\mathsf{Sign}} \le t$ and $|\mathcal{L}| \le 3t + 2$, we can apply Lemma 1 to conclude that

$$\begin{aligned}\Pr[\mathsf{SigForge}^{\mathtt{GO}}_{\mathcal{A}_{\mathsf{sig}},\Pi}(k) = 1] &\le \frac{3t^2 + 3t + 3}{q - (3t + 2)^2} + \frac{t^2}{q} + \frac{t}{q - 3t - 2} + \frac{t + 1}{2^k}\\ &\le \frac{3t^2 + 4t + 3}{q - (3t + 2)^2} + \frac{t^2}{q} + \frac{t + 1}{2^k} = \mathcal{O}\left(\frac{t}{2^k}\right)\ .\end{aligned}$$

Note that for all $t < \frac{2^{k-1} - 2}{3}$ we have that $(3t + 2)^2 < \frac{2^{2k}}{4} \approx \frac{q}{4}$ and therefore it clearly holds that $\frac{3t^2+4t+3}{q-(3t+2)^2} = \mathcal{O}\left(\frac{t}{2^k}\right)$. □

4 Multi-User Security of Short Schnorr Signatures

We are able to extend the previous security proof of *short* Schnorr signatures to the multi-user security with the similar reduction. We begin by introducing the 1-out-of-N signature forgery game in the generic group plus random oracle model. The attacker is given N independent public keys $pk_1, \ldots, pk_N = \tau(g^{sk_1}), \ldots, \tau(g^{sk_N})$ along with oracle access to the signing oracles $\mathsf{Sign}(sk_1, \cdot), \ldots, \mathsf{Sign}(sk_N, \cdot)$. The attacker can succeed if he can output a forgery (σ, m) which is valid under *any* public key e.g., for some public key pk_j we have $\mathsf{Vfy}(pk_j, m, \sigma) = 1$ while the query m was never submitted to the jth signing oracle $\mathsf{Sign}(sk_j, \cdot)$.

In this sense, we argue that the *short* Schnorr signatures have the multi-user security in the "1-out-of-N" setting which implies that the probability that the attacker can forge any one of N signatures is negligible. We first define the following security games in this multi-user setting.

1-out-of-N Generic Signature Forgery Game. Consider the generic group model of prime order $q \approx 2^{2k}$ as stated before. Consider the following experiment defined for a signature scheme $\Pi = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vfy})$, the generic oracles $\mathtt{GO} = (\mathtt{Mult}(\cdot,\cdot), \mathtt{Inv}(\cdot))$, and adversary $\mathcal{A}$:

The 1-out-of-N Generic Signature Forgery Game $\mathsf{SigForge}^{\mathtt{GO},N}_{\mathcal{A},\Pi}(k)$:

(1) $\mathsf{Kg}(1^k)$ is run N times to obtain the public and the secret keys (pk_i, sk_i) for $1 \le i \le N$. Here, for each $1 \le i \le N$, sk_i is chosen randomly from the group $G = \mathbb{Z}_q$ where q is a $2k$-bit prime, and $pk_i = \tau(g^{sk_i})$ where g is the generator of the group G.
(2) Adversary $\mathcal{A}$ is given $(\mathfrak{g} = \tau(g), pk_1, \cdots, pk_N, q)$ and access to the generic group oracles $\mathtt{GO} = (\mathtt{Mult}(\cdot,\cdot), \mathtt{Inv}(\cdot))$, the random oracle $\mathsf{H}(\cdot)$ and the signing oracles $\mathsf{Sign}(sk_1, \cdot), \ldots, \mathsf{Sign}(sk_N, \cdot)$. After multiple access to these oracles, the adversary outputs $(m, \sigma = (s, e))$.
(3) $\mathcal{A}$ succeeds to forge a signature if and only if there exists $1 \le j \le N$ such that $\mathsf{Vfy}(pk_j, m, \sigma) = 1$ and the query m was never submitted to the oracle $\mathsf{Sign}(sk_j, \cdot)$. We define that $\mathsf{SigForge}^{\mathtt{GO},N}_{\mathcal{A},\Pi}(k) = 1$ when $\mathcal{A}$ succeeds.

Definition 2. *We say that a signature scheme $\Pi = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vfy})$ is* $(t, N, q_{\mathtt{RO}}, q_{\mathtt{GO}}, q_{\mathsf{Sign}}, \epsilon)$-MU-UF-CMA secure (multi-user unforgeable against chosen message attack) *if for every adversary $\mathcal{A}$ running in time at most t and making at most $q_{\mathtt{RO}}$ (resp. $q_{\mathtt{GO}}$, q_{Sign}) queries to the random oracle (resp. generic group, signature oracles), the following bound holds:*

$$\Pr\left[\mathsf{SigForge}^{\mathtt{GO},N}_{\mathcal{A},\Pi}(k) = 1\right] \le \epsilon.$$

The Known/Partially Known Set in a Multi-User Setting. As before we will maintain the invariant that for every output $\mathfrak{y}$ of a generic group query that we have recorded a tuple $(\mathfrak{y}, \boldsymbol{a}, b)$ in a list $\mathcal{L}$ where $\mathtt{DLog}_g(\mathfrak{y}) = \boldsymbol{a} \cdot \boldsymbol{x} + b$ (here, $\boldsymbol{a}$ and $\boldsymbol{x} = (x_1, \ldots, x_N)$ are N-dimensional vectors). We can partition the global list $\mathcal{L}$ into a known set $\mathcal{K}^N$[3] which contains tuples of the form $(\mathfrak{y}, \mathbf{0}, b)$ (discrete log of τ is known to be b) and a partially known set $\mathcal{PK}^N_{\{x_i\}_{i=1}^N}$ which contains tuples of the form $(\mathfrak{y}, \boldsymbol{a}, b)$ with $\boldsymbol{a} \ne \mathbf{0}$. We can maintain the invariant that every output $\mathfrak{y}$ is contained in $\mathcal{L}$ using our restricted oracle $\mathtt{DLog}_g(\cdot)$ which will solve $\mathtt{DLog}_g(\mathfrak{y})$ for any fresh group element $\mathfrak{y}$ such that $\mathfrak{y} \notin \{\tau(g^{x_1}), \ldots, \tau(g^{x_N})\}$ and $\mathfrak{y}$ has not been the output of a prior generic group query.

- Initially, $\mathcal{K}^N$ contains $(\tau(g), \mathbf{0}, 1)$ and $\mathcal{PK}^N_{\{x_i\}_{i=1}^N}$ contains $(\tau(h_i) = \tau(g^{x_i}), \boldsymbol{e}_i, 0)$ for $1 \le i \le N$, where $\boldsymbol{e}_i$ is the vector such that $\boldsymbol{e}_i[i] = 1$ and $\boldsymbol{e}_i[j] = 0$ for all $j \ne i$.

[3] To avoid confusion, we remark that the superscript N is used to denote the known set in the 1-out-of-N multi-user setting. That is, $\mathcal{K}^N \ne \mathcal{K} \times \cdots \times \mathcal{K}$. Similar for the partially known sets.

- If the attacker every submits a fresh group element $\mathfrak{y}$ which was not previously the output of a group query then we can query $b = \texttt{DLog}_g(\mathfrak{y})$ and add $(\mathfrak{y}, \mathbf{0}, b)$ to our list. Thus, without loss of generality, we can assume that all query inputs to $\texttt{Mult}, \texttt{Inv}$ were first added to $\mathcal{L}$.
- If $(\mathfrak{y}_1, \boldsymbol{a}_1, b_1), (\mathfrak{y}_2, \boldsymbol{a}_2, b_2) \in \mathcal{L}$ and the attacker queries $\texttt{Mult}(\mathfrak{y}_1, \mathfrak{y}_2)$ we will add $(\texttt{Mult}(\mathfrak{y}_1, \mathfrak{y}_2), \boldsymbol{a}_1 + \boldsymbol{a}_2, b_1 + b_2)$ to $\mathcal{L}$.
- If $(\mathfrak{y}, \boldsymbol{a}, b) \in \mathcal{L}$ and the attacker queries $\texttt{Inv}(\mathfrak{y})$ we will add $(\texttt{Inv}(\mathfrak{y}), -\boldsymbol{a}, -b)$ to $\mathcal{L}$.

4.1 The Multi-User Bridge Game

We establish the multi-user security of short Schnorr signatures via reduction from a new game we introduce called the "1-out-of-N generic BRIDGE^N-finding game". As in the 1-out-of-N discrete-log game the attacker is given $\tau(g)$ as well as $\tau(g^{x_1}), \ldots, \tau(g^{x_N})$ for N randomly selected values $x_1, \ldots, x_N \in \mathbb{Z}_q$. The key difference between this game and the (multi) discrete-log problem is that the attackers goal is simply to ensure that the "bridge event" BRIDGE^N occurs whether or not the attacker is able to solve any of the discrete-log challenges. As in the previous sections we will assume that we have access to a restricted discrete-log oracle and we will maintain the invariant that for every output $\tau(y)$ of some generic group query we have $y = g^{b + \sum_i a_i x_i}$ for known values $b, a_1, \ldots, a_N \in \mathbb{Z}_q$ i.e., by querying the restricted oracle $\texttt{DLog}_g(\tau(y))$ whenever we encounter a fresh input. As before we can introduce the known/partially known sets $\mathcal{K}^N$ and $\mathcal{PK}^N_{\{x_i\}_{i=1}^N}$ i.e., depending on whether $(a_1, \ldots, a_N) = \mathbf{0}$ or not.

The event BRIDGE^N occurs if $\mathcal{L}$ ever contains two distinct tuples $(\mathfrak{y}_1, \boldsymbol{a}_1, b_1)$ and $(\mathfrak{y}_2, \boldsymbol{a}_2, b_2)$ such that $\mathfrak{y}_1 = \mathfrak{y}_2$ but $(\boldsymbol{a}_1, b_1) \neq (\boldsymbol{a}_2, b_2)$. Intuitively, if this occurs then the attacker can "bridge" the sets $\mathcal{K}^N$ and $\mathcal{PK}^N_{\{x_i\}_{i=1}^N}$ as follows: 1) compute $\tau(g^{b_2 - b_1})$ using b_1 and b_2 to ensure $(\tau(g^{b_2 - b_1}), \mathbf{0}, b_2 - b_1) \in \mathcal{K}^N$, and 2) compute $\tau(g^{\boldsymbol{c} \cdot \boldsymbol{x}}) = \tau(\prod_i h_i^{\boldsymbol{c}[i]})$ using $\boldsymbol{c} = \boldsymbol{a}_1 - \boldsymbol{a}_2$ to ensure that $\tau(\tau(g^{b_2 - b_1}), \boldsymbol{c}, 0) \in \mathcal{PK}^N_{\{x_i\}_{i=1}^N}$ — note that $b_2 - b_1 = \boldsymbol{c} \cdot \boldsymbol{x}$. As long as the event BRIDGE^N has not occurred we can (essentially) view $x_1, \ldots, x_N$ as uniformly random values that that yet to be selected. More precisely, the values $x_1, \ldots, x_N$ are selected subject to a few constraints, e.g., if we know $\mathfrak{f}_1 = \tau(g^{\boldsymbol{a}_1 \cdot \boldsymbol{x} + b_1}) \neq \mathfrak{f}_2 = \tau(g^{\boldsymbol{a}_2 \cdot \boldsymbol{x} + b_2})$ then we have the constraint that $\boldsymbol{a}_1 \cdot \boldsymbol{x} + b_1 \neq \boldsymbol{a}_2 \cdot \boldsymbol{x} + b_2$.

The 1-out-of-N Generic BRIDGE^N-Finding Game $\mathsf{BridgeChal}^{\texttt{GO},N}_{\mathcal{A}}(k)$:

(1) The challenger selects uniformly random $x_1, \cdots, x_N \in \mathbb{Z}_q^n$ and initializes the list $\mathcal{L} = ((\tau(g), \mathbf{0}, 1), (\tau(g^{x_1}), \boldsymbol{e}_1, 0), \ldots, (\tau(g^{x_N}), \boldsymbol{e}_N, 0))$ where $\boldsymbol{e}_i$ is the vector such that $\boldsymbol{e}_i[i] = 1$ and $\boldsymbol{e}_i[j] = 0$ for all $j \neq i$.

(2) The adversary $\mathcal{A}$ is given $\mathfrak{g} = \tau(g)$, $\tau(h_i) = \tau(g^{x_i}), 1 \leq i \leq N$ for the random values of $x_1, \cdots, x_N$. Here, g is a generator of a cyclic group $G = \langle g \rangle$ of prime order $q \approx 2^{2k}$ and $\tau : G \to \mathbb{G}$ is the map from G to a generic group $\mathbb{G}$.

(3) $\mathcal{A}$ is allowed to query the usual generic group oracles $(\texttt{Mult}, \texttt{Inv})$. If the challenger ever submits any fresh element $\mathfrak{y}$ which does not appear in $\mathcal{L}$ as input to a generic group oracle then the challenger immediately queries $b_y = \texttt{DLog}_g(\mathfrak{y})$ and adds the tuple $(\mathfrak{y}, \mathbf{0}, b_y)$ to $\mathcal{L}$. Whenever $\mathcal{A}$ submits a query $\mathfrak{y}$ to $\texttt{Inv}(\cdot)$ we are ensured that some tuple $(\mathfrak{y}, \boldsymbol{a}_y, b_y) \in \mathcal{L}$. The challenger adds the tuple $(\texttt{Inv}(\mathfrak{y}), -\boldsymbol{a}_y, -b_y)$ to $\mathcal{L}$. Whenever $\mathcal{A}$ submits a query $\mathfrak{y}_1, \mathfrak{y}_2$ to $\texttt{Mult}(\cdot, \cdot)$ we are ensure that there exist tuples $(\mathfrak{y}_1, \boldsymbol{a}_1, b_1), (\mathfrak{y}_2, \boldsymbol{a}_2, b_2) \in \mathcal{L}$. The challenger adds the tuple $(\texttt{Mult}(\mathfrak{y}_1, \mathfrak{y}_2), \boldsymbol{a}_1 + \boldsymbol{a}_2, b_1 + b_2)$ to $\mathcal{L}$.

(4) If at any point in time we have a collision, i.e., two distinct tuples $(\mathfrak{y}, \boldsymbol{a}_1, b_1), (\mathfrak{y}, \boldsymbol{a}_1, b_1) \in \mathcal{L}$ with $(\boldsymbol{a}_1, b_1) \neq (\boldsymbol{a}_2, b_2)$ then the event BRIDGE^N occurs and the output of the game is 1 (bridge attacker succeeds). If BRIDGE^N never occurs then the output of the game is 0 (bridge attacker fails).

Theorem 4. *The probability an attacker $\mathcal{A}$ running in time t wins the 1-out-of-N generic BRIDGE^N-finding game (even with access to the restricted* $\texttt{DLog}$ *oracle) is at most*

$$\Pr\left[\mathsf{BridgeChal}_{\mathcal{A}}^{\texttt{GO},N}(k) = 1\right] \leq \frac{tN + 3t(t+1)/2}{q - (N + 3t + 1)^2 - N}$$

where q is the order of the group G.

Proof. Consider the output $\mathfrak{y}_i$ of the i^{th} generic group query. We first analyze the probability that this query results in the event BRIDGE^N conditioning on the event $\overline{\mathsf{BRIDGE}}_{<i}^N$ that the event has not yet occurred. Before we even receive the output $\mathfrak{y}_i$ we already know the values $\boldsymbol{a}_i, b_i$ s.t. the tuple $(\mathfrak{y}_i, \boldsymbol{a}_i, b_i)$ will be added to $\mathcal{L}$. If $\mathcal{L}$ does already contain this *exact* tuple then outputting $\mathfrak{y}_i$ will not produce the event BRIDGE^N. If $\mathcal{L}$ does not already contain this tuple $(\mathfrak{y}_i, \boldsymbol{a}_i, b_i)$ then we are interested in the event B_i that some other tuple $(\mathfrak{y}_i, \boldsymbol{a}_i', b_i')$ has been recorded with $(\boldsymbol{a}_i', b_i') \neq (\boldsymbol{a}_i, b_i)$. Observe that B_i occurs if and only if there exists a tuple of the form $(\cdot, \boldsymbol{a}, b)$ with $(\boldsymbol{a} - \boldsymbol{a}_i) \cdot \boldsymbol{x} = b_i - b$ and $(\boldsymbol{a}, b) \neq (\boldsymbol{a}_i, b_i)$. If we pick $\boldsymbol{x}$ randomly the probability that $(\boldsymbol{a} - \boldsymbol{a}_i) \cdot \boldsymbol{x} = b_i - b$ would be $1/q$. However, we cannot quite view $\boldsymbol{x}$ as random due to the restrictions, i.e., because we condition of the event $\overline{\mathsf{BRIDGE}}_{<i}^N$ we know that for any distinct pair $(\mathfrak{y}_i, \boldsymbol{a}_i, b_i)$ and $(\mathfrak{y}_i, \boldsymbol{a}_j, b_j)$ we know that $\boldsymbol{a}_i \cdot \boldsymbol{x} + b_i \neq \boldsymbol{a}_j \cdot \boldsymbol{x} + b_j$.

Consider sampling $\boldsymbol{x}$ uniformly at random subject to this restriction. Let $r \leq N$ be an index s.t. $\boldsymbol{a}[r] - \boldsymbol{a}_i[r] \neq 0$ and suppose that $x_r = \boldsymbol{x}[r]$ is the last value sampled. At this point we can view x_r as being drawn uniformly at random from a set of at least $q - |\mathcal{L}|^2 - (N-1)$ remaining values subject to all of the restrictions. We also observe that $|\mathcal{L}| \leq N + 3t + 1$ since each generic group oracle adds *at most* three new tuples to $\mathcal{L}$ — exactly three in the case that we where $\texttt{Mult}(\mathfrak{y}_1, \mathfrak{y}_2)$ on two fresh elements. Thus, the probability that $(\boldsymbol{a} - \boldsymbol{a}_i) \cdot \boldsymbol{x} = b_i - b$ is at most $\frac{1}{q - (N+3t+1)^2 - (N-1)}$. Union bounding over all tuples $(\cdot, \boldsymbol{a}, b) \in \mathcal{L}$ we have

$$\Pr\left[B_i : \overline{\mathsf{BRIDGE}}^N_{<i}\right] \le \frac{N+3i}{q-(N+3t+1)^2-N} \ .$$

To complete the proof we observe that

$$\begin{aligned}\Pr\left[\mathsf{BridgeChal}^{\mathtt{GO},N}_{\mathcal{A}}(k)=1\right] &= \sum_{i\le t}\Pr\left[B_i : \overline{\mathsf{BRIDGE}}^N_{<i}\right] \\ &\le \sum_{i\le t}\frac{N+3i}{q-(N+3t+1)^2-N} \\ &= \frac{tN+3t(t+1)/2}{q-(N+3t+1)^2-N} \ .\end{aligned}$$

□

As an immediate corollary of Theorem 4 we can show that an attacker wins the 1-out-of-N discrete log game with (approximately) the same probability as in the bridge game. In particular, given any attacker $\mathcal{A}'$ in the discrete-log-game $\mathsf{1ofNDLog}^{\mathtt{GO},N}_{\mathcal{A}}(k)$ where the attackers goal is to output *any* $x \in \{x_1,\ldots,x_N\}$ given input $\tau(g),\tau(g^{x_1}),\ldots,\tau(g^{x_n})$ we can construct an attacker $\mathcal{A}$ in the game $\mathsf{BridgeChal}^{\mathtt{GO},N}_{\mathcal{A}}(k)$. $\mathcal{A}$ simply runs $\mathcal{A}'$ to obtain an output x and then computes $\tau(g^x)$ using at most $2\log q$ queries to the $\mathtt{Mult}(\cdot,\cdot)$ oracle. If $x\in\{x_1,\ldots,x_N\}$ then the bridge event BRIDGE^N must have occurred at some point since we have $(\tau(g^x),\mathbf{0},x)\in\mathcal{L}$ and $(\tau(g^x),\boldsymbol{e}_i,0)\in\mathcal{L}$ for some $i\le N$.

Corollary 1. *For any attacker $\mathcal{A}$ running in time $t' = t+2\log q$ we have*

$$\Pr\left[\mathsf{1ofNDLog}^{\mathtt{GO},N}_{\mathcal{A}}(k)=1\right] \le \frac{tN+3t(t+1)/2}{q-(N+3t+1)^2-N}$$

where q is the order of the group G.

4.2 Security Reduction

Theorem 5. *In the generic group model of prime order $q\approx 2^{2k}$ and the programmable random oracle model the short Schnorr signature scheme is* $(t,N,q_{\mathtt{RO}},q_{\mathtt{GO}},q_{\mathsf{Sign}},\epsilon)$-MU-UF-CMA *secure with* $\epsilon = \frac{tN+3t(t+2)/2}{q-(N+3t+1)^2-N}+\frac{t^2}{q}+\frac{t+1}{2^k} = \mathcal{O}\left(\frac{t+N}{2^k}\right)$.

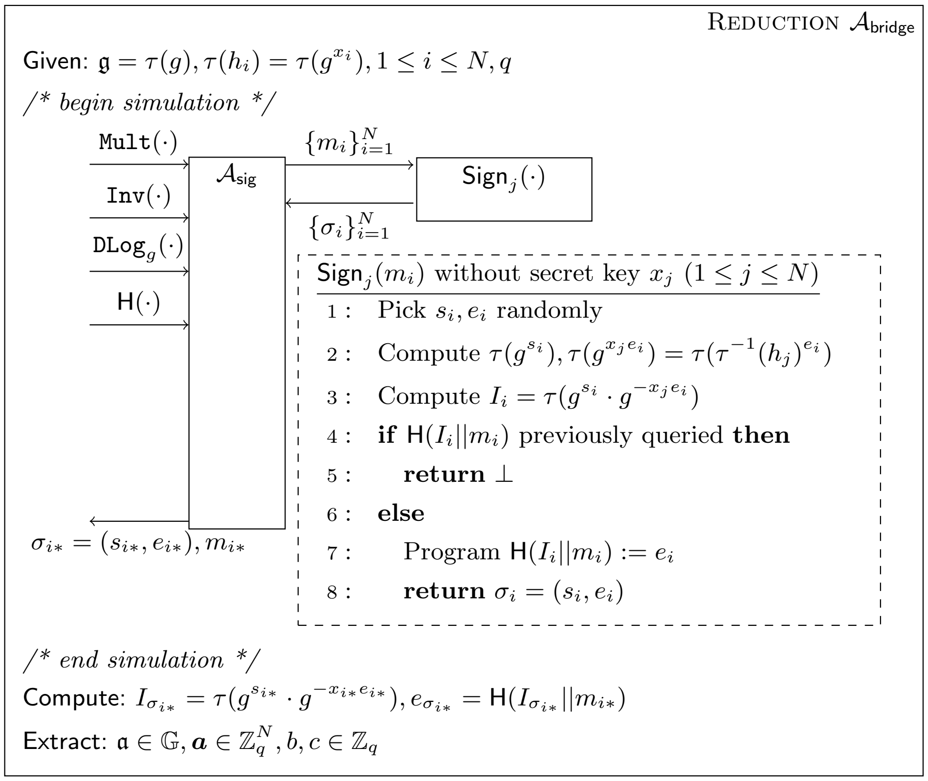
Proof. Let $\mathit{\Pi} = (\mathsf{Kg},\mathsf{Sign},\mathsf{Vfy})$ be the Schnorr signature scheme. Given an adversary $\mathcal{A}_{\mathsf{sig}}$ attacking Schnorr signature scheme, we construct the following efficient algorithm $\mathcal{A}_{\mathsf{bridge}}$ which tries to succeed in the 1-out-of-N generic BRIDGE^N-finding game $\mathsf{BridgeChal}^{\mathtt{GO},N}_{\mathcal{A}_{\mathsf{bridge}}}(k)$:

Algorithm $\mathcal{A}_{\mathsf{bridge}}$:
The algorithm is given $\mathfrak{g}=\tau(g),\tau(h_i)=\tau(g^{x_i}),1\le i\le N,q$ as input.

1. Initialize the list $\mathcal{L} = \{(\tau(g), \mathbf{0}, 1), (\tau(h_i) = \tau(g^{x_i}), \boldsymbol{e}_i, 0) \text{ for } 1 \leq i \leq N\}$ where $\boldsymbol{e}_i$ is the vector such that $\boldsymbol{e}_i[i] = 1$ and $\boldsymbol{e}_i[j] = 0$ for all $j \neq i$, and $H_{\mathsf{resp}} = \{\}$ where H_{resp} stores the random oracle queries.
2. Run $\mathcal{A}_{\mathsf{sig}}$ with a number of access to the generic oracles $\mathtt{GO} = (\mathtt{Mult}(\cdot,\cdot), \mathtt{Inv}(\cdot)), \mathtt{DLog}_g(\cdot), \mathsf{Sign}_i(\cdot)$ for $1 \leq i \leq N$, and $\mathsf{H}(\cdot)$. The signing oracle without a secret key is described in Figure 3. Now we consider the following cases:
   (a) Whenever $\mathcal{A}_{\mathsf{sig}}$ submits a query x_i to the random oracle H:
   – If there is a pair $(x_i, r_i) \in H_{\mathsf{resp}}$ for some string r_i then return r_i.
   – Otherwise, select $r_i \in \mathbb{Z}_{2^k}$ uniformly at random and add (x_i, r_i) to the set H_{resp}.
   – If x_i has the form $x_i = (\mathfrak{a}_i || m_i)$ where the value $\mathfrak{a}_i$ has not been observed previously (i.e., is not in the list $\mathcal{L}$) then we query $b_i = \mathtt{DLog}_g(\mathfrak{a}_i)$ and add $\big(\tau(g^{b_i}), \mathbf{0}, b_i\big)$ to $\mathcal{L}$.
   (b) Whenever $\mathcal{A}_{\mathsf{sig}}$ submits a query $\mathfrak{a}$ to the generic group oracle $\mathtt{Inv}(\mathfrak{a})$:
   – If $\mathfrak{a}$ is not in the list $\mathcal{L}$ then we immediately query $b = \mathtt{DLog}_g(\mathfrak{a})$ and add $\big(\tau(g^{b}), \mathbf{0}, b\big)$ to $\mathcal{L}$.
   – Otherwise, $(\mathfrak{a}, \boldsymbol{a}, b) \in \mathcal{L}$. Then we query $\mathtt{Inv}(\mathfrak{a}) = \tau(g^{-\boldsymbol{a}\cdot\boldsymbol{x}-b})$, output the result and add the result $\big(\tau(g^{-\boldsymbol{a}\cdot\boldsymbol{x}-b}), -\boldsymbol{a}, -b\big) \in \mathcal{L}$.
   (c) Whenever $\mathcal{A}_{\mathsf{sig}}$ submits a query $\mathfrak{a}, \mathfrak{b}$ to the generic group oracle $\mathtt{Mult}(\mathfrak{a}, \mathfrak{b})$:
   – If the element $\mathfrak{a}$ (resp. $\mathfrak{b}$) is not in $\mathcal{L}$ then query $b_0 = \mathtt{DLog}_g(\mathfrak{a})$ (resp. $b_1 = \mathtt{DLog}_g(\mathfrak{b})$) and add the element $(\mathfrak{a}, \mathbf{0}, b_0)$ (resp. $(\mathfrak{b}, \mathbf{0}, b_1)$) to $\mathcal{L}$.
   – Otherwise both elements $(\mathfrak{a}, \boldsymbol{a}_0, b_0), (\mathfrak{b}, \boldsymbol{a}_1, b_1) \in \mathcal{L}$. Then we return $\mathtt{Mult}(\mathfrak{a}, \mathfrak{b}) = \tau(g^{(\boldsymbol{a}_0+\boldsymbol{a}_1)\cdot\boldsymbol{x}+b_0+b_1})$ and add the element $(\tau(g^{(\boldsymbol{a}_0+\boldsymbol{a}_1)\cdot\boldsymbol{x}+b_0+b_1}), \boldsymbol{a}_0 + \boldsymbol{a}_1, b_0 + b_1) \in \mathcal{L}$.
   (d) Whenever $\mathcal{A}_{\mathsf{sig}}$ submits a query m_i to the signing oracle $\mathsf{Sign}(x_j, \cdot)$:
   – We use the procedure Sign_j described in Figure 3 to forge a signature without knowledge of the secret key x_i. Intuitively, the forgery procedure relies on our ability to program the random oracle.
   – We remark that a side effect of querying the Sign_j oracle is the addition of the triples $(\tau(g^{s_i}), \mathbf{0}, s_i)$, $(\tau(g^{x_j e_i}), \boldsymbol{e}_i^{[i]}, 0)$ and $(\tau(g^{s_i - x_j e_i}), -\boldsymbol{e}_i^{[i]}, s_i)$ to $\mathcal{L}$ where $\boldsymbol{e}_i^{[i]} = (0, \cdots 0, e_i, 0 \cdots, 0) \in \mathbb{Z}_q^N$ denotes the vector with the i^{th} element e_i and 0 elsewhere, since these values are computed using the generic group oracles $\mathtt{Inv}, \mathtt{Mult}$.
   (e) If at any point we have some string $\mathfrak{y}$ such that $(\mathfrak{y}, \boldsymbol{a}, b) \in \mathcal{L}$ and $(\mathfrak{y}, \boldsymbol{c}, d) \in \mathcal{L}$ for $(\boldsymbol{a}, b) \neq (\boldsymbol{c}, d)$ then we can immediately have a BRIDGE^N instance $(\tau(g^{(\boldsymbol{a}-\boldsymbol{c})\cdot\boldsymbol{x}}), \boldsymbol{a} - \boldsymbol{c}, 0) \in \mathcal{L}$ and $(\tau(g^{d-b}), \mathbf{0}, d - b) \in \mathcal{L}$ since $\tau(g^{(\boldsymbol{a}-\boldsymbol{c})\cdot\boldsymbol{x}}) = \tau(g^{d-b})$.[4] Thus, without loss of generality, we can assume that each string $\mathfrak{y}$ occurs at most once in the list $\mathcal{L}$.
3. After $\mathcal{A}_{\mathsf{sig}}$ outputs $\sigma_{i*} = (s_{i*}, e_{i*})$ and m_{i*} we first compute $I_{\sigma_{i*}} = \mathtt{Mult}(\mathtt{Pow}(\tau(g), s_{i*}), \mathtt{Inv}(\mathtt{Pow}(h_{i*}, e_{i*}))) = \tau(g^{s_{i*}} \cdot g^{-x_{i*}e_{i*}})$ and then check to see if we previously had any triple of the form $(I_{\sigma_{i*}}, \boldsymbol{a}, b) \in \mathcal{L}$.
   (a) If no such tuple exists we return $\perp$.
   (b) Otherwise, we let $\boldsymbol{a}, b$ be given such that $I_{\sigma_{i*}} = \tau(g^{\boldsymbol{a}\cdot\boldsymbol{x}+b})$.

[4] Note that $(\boldsymbol{a}, b) \neq (\boldsymbol{c}, d)$ implies $\boldsymbol{a} \neq \boldsymbol{c}$ since if $\boldsymbol{a} = \boldsymbol{c}$ then $g^{\boldsymbol{a}\cdot\boldsymbol{x}+b} = g^{\boldsymbol{a}\cdot\boldsymbol{x}+d}$ implies $b = d$ as $b, d \in \mathbb{Z}_q$.

– If $\boldsymbol{a} + \boldsymbol{e}_{i*}^{[i*]} = 0$ then we return $\perp$ where $\boldsymbol{e}_{i*}^{[i*]} = (0, \cdots 0, e_{i*}, 0 \cdots, 0) \in \mathbb{Z}_q^N$ denotes the vector with the $i*^{th}$ element e_{i*} and 0 elsewhere.
– Otherwise, we return a BRIDGE^N instance $(\tau(g^{s_{i*}-b}), \boldsymbol{a} + \boldsymbol{e}_{i*}^{[i*]}, 0) \in \mathcal{L}$ and $(\tau(g^{s_{i*}-b}), \mathbf{0}, s_{i*} - b) \in \mathcal{L}$.



Given: $\mathfrak{g} = \tau(g), \tau(h_i) = \tau(g^{x_i}), 1 \le i \le N, q$

/ begin simulation */*

```
Sign_j(m_i) without secret key x_j (1 ≤ j ≤ N)
1 : Pick $s_i, e_i$ randomly
2 : Compute $\tau(g^{s_i}), \tau(g^{x_j e_i}) = \tau(\tau^{-1}(h_j)^{e_i})$
3 : Compute $I_i = \tau(g^{s_i} \cdot g^{-x_j e_i})$
4 : if $\mathsf{H}(I_i || m_i)$ previously queried then
5 :     return $\perp$
6 : else
7 :     Program $\mathsf{H}(I_i || m_i) := e_i$
8 :     return $\sigma_i = (s_i, e_i)$
```

/ end simulation */*

Compute: $I_{\sigma_{i*}} = \tau(g^{s_{i*}} \cdot g^{-x_{i*} e_{i*}}), e_{\sigma_{i*}} = \mathsf{H}(I_{\sigma_{i*}} || m_{i*})$

Extract: $\mathfrak{a} \in \mathbb{G}, \boldsymbol{a} \in \mathbb{Z}_q^N, b, c \in \mathbb{Z}_q$

Figure 3. A reduction to the $\mathsf{BridgeChal}_{\mathcal{A}_{\mathsf{bridge}}}^{\mathtt{GO},N}(k)$ attacker $\mathcal{A}_{\mathsf{bridge}}$ from the Schnorr signature attacker $\mathcal{A}_{\mathsf{sig}}$.

Analysis. We first remark that if the signature is valid then we must have $e_{i*} = \mathsf{H}(I_{\sigma_{i*}} || m_{i*})$ and $\mathtt{DLog}_g(I_{\sigma_{i*}}) = s_{i*} - x_{i*} e_{i*} = \boldsymbol{a} \cdot \boldsymbol{x} + b$ or equivalently $s_{i*} - b = (\boldsymbol{a} + \boldsymbol{e}_{i*}^{[i*]}) \cdot \boldsymbol{x}$. Thus, as long as $\boldsymbol{a} + \boldsymbol{e}_{i*}^{[i*]}$ is a nonzero vector we will have a BRIDGE^N instance $(\tau(g^{s_{i*}-b}), \boldsymbol{a} + \boldsymbol{e}_{i*}^{[i*]}, 0) \in \mathcal{L}$ and $(\tau(g^{s_{i*}-b}), \mathbf{0}, s_{i*} - b) \in \mathcal{L}$.

Now consider the upper bound of the probability that our algorithm outputs $\perp$ for failure before $\mathcal{A}_{sig}$ outputs a signature as well as the probability our algorithm outputs $\perp$ after $\mathcal{A}_{sig}$ outputs a valid signature.

(1) One way to output failure is during the signing oracle if $\mathsf{H}(I_i || m_i)$ has been queried previously (Algorithm 2.(d)). We define this event as $\mathsf{FailtoSign}^N$.

Then we have the following claim to upper bound the probability of the event:

Claim 4. $\Pr[\mathsf{FailtoSign}^N] \leq q_{\mathsf{Sign}} \times \dfrac{q_{\mathtt{RO}} + q_{\mathsf{Sign}}}{q}$.

Proof. Since, $I_i = \mathtt{Pow}(\tau(g), r_i)$ represents a fresh/randomly selected group element we can use union bounds to upper bound the probability that this failure event ever occurs as $\Pr[\mathsf{FailtoSign}^N] \leq q_{\mathsf{Sign}} \times \frac{q_{\mathtt{RO}}+q_{\mathsf{Sign}}}{q}$ where q_{Sign} is the total number of queries to the signing oracle and $q_{\mathtt{RO}}$ is the total number of queries to the random oracle. $\square$

(2) The second way to output failure is if the value $I_{\sigma_{i*}}$ does not previously appear in the list $\mathcal{L}$ (Algorithm 3.(a)). We use $\mathsf{FailtoFind}^N(I_{\sigma_{i*}})$ to denote the event that the signature is valid but the value $I_{\sigma_{i*}}$ does not appear in $\mathcal{L}$.

Claim 5. $\Pr[\mathsf{FailtoFind}^N(I_{\sigma_{i*}})] \leq \dfrac{q_{\mathtt{RO}} + q_{\mathsf{Sign}}}{q - |\mathcal{L}|} + \dfrac{1}{2^k}$.

Proof. If $I_{\sigma_{i*}} \notin \mathcal{L}$ then we can view $I_{\sigma_{i*}} = \tau(g^{s_{i*}} \cdot g^{-x_{i*}e_{i*}})$ as a uniformly random binary string from a set of size at least $q - |\mathcal{L}|$ which had not yet been selected at the time $\mathcal{A}_{\mathsf{sig}}$ output σ_{i*}. Thus, the probability that the query $\mathsf{H}(I_{\sigma_{i*}}||m_{i*})$ was previously recorded is at most $(q_{\mathtt{RO}} + q_{\mathsf{Sign}})/\,(q - |\mathcal{L}|)$. Observe that if the query $\mathsf{H}(I_{\sigma_{i*}}||m_{i*})$ *was not* previously recorded then the probability of a successful forgery $\mathsf{H}(I_{\sigma_{i*}}||m_{i*}) = e_{i*}$ is *at most* 2^{-k} since we can view $\mathsf{H}(I_{\sigma_{i*}}||m_{i*})$ as a uniformly random k-bit string. Hence, we have that $\Pr[\mathsf{FailtoFind}^N(I_{\sigma_{i*}})] \leq \frac{q_{\mathtt{RO}}+q_{\mathsf{Sign}}}{q-|\mathcal{L}|} + \frac{1}{2^k}$. $\square$

(3) Finally, we could output failure if $\boldsymbol{a} = -\boldsymbol{e}_{i*}^{[i*]}$ (Algorithm 3.(b)). We call this event $\mathsf{BadQuery}^N$.

Claim 6. $\Pr[\mathsf{BadQuery}^N] \leq \dfrac{q_{\mathtt{RO}}}{2^k}$.

Proof. We first note that by construction we ensure that the tuple $(I, \boldsymbol{a}, b)$ will always be recorded in $\mathcal{L}$ before a query of the form $\mathsf{H}(I||m)$ is ever issued — if I is new then we call $\mathtt{DLog}_g(I)$ before querying the random oracle. Now define a subset $\widehat{\mathcal{L}} \subset \mathcal{L}$ as the set of tuples $(\mathfrak{a}, \boldsymbol{a}, b) \in \mathbb{G} \times \mathbb{Z}_q^N \times \mathbb{Z}_q \subset \mathcal{L}$ such that $\boldsymbol{a}$ has exactly *one* nonzero element. Now we call a random oracle query $x = I||m$ "bad" if $\mathsf{H}(x) = \widetilde{a}$ where the triple $(I, \boldsymbol{a}, b) \in \widehat{\mathcal{L}}$ has already been recorded and the nonzero element of $\boldsymbol{a}$ is $\widetilde{a}$ (Recall that if there were two recorded triples $(I, \boldsymbol{a}, b)$ and $(I, \boldsymbol{c}, d)$ then our algorithm would have already found a BRIDGE^N instance). We can use union bounds to upper bound the probability of any "bad" query as $\Pr[\mathsf{BadQuery}^N] \leq \frac{q_{\mathtt{RO}}}{2^k}$. $\square$

Now We have shown that

$$\Pr[\mathsf{BridgeChal}^{\mathtt{GO},N}_{\mathcal{A}_{\mathsf{bridge}}}(k) = 1] \geq \Pr[\mathsf{SigForge}^{\mathtt{GO},N}_{\mathcal{A}_{\mathsf{sig}},\Pi}(k) = 1] - \Pr[\mathsf{FailtoSign}^N]$$

$$
\begin{aligned}
&- \Pr[\mathsf{FailtoFind}^N(I_{\sigma_{i*}})] - \Pr[\mathsf{BadQuery}^N] \\
&\geq \Pr[\mathsf{SigForge}^{\mathtt{GO},N}_{\mathcal{A}_{\mathsf{sig}},\Pi}(k)] - \frac{q_{\mathsf{Sign}}(q_{\mathtt{RO}} + q_{\mathsf{Sign}})}{q} \\
&- \frac{q_{\mathtt{RO}} + q_{\mathsf{Sign}}}{q - |\mathcal{L}|} - \frac{1}{2^k} - \frac{q_{\mathtt{RO}}}{2^k} .
\end{aligned}
$$

Since we have $|\mathcal{L}| \leq N + 3t + 1$ and $q_{\mathtt{RO}} + q_{\mathsf{Sign}} \leq t$, we can apply Theorem 4 to conclude that

$$
\begin{aligned}
\Pr[\mathsf{SigForge}^{\mathtt{GO},N}_{\mathcal{A}_{\mathsf{sig}},\Pi}(k) = 1] &\leq \tfrac{tN+3t(t+1)/2}{q-(N+3t+1)^2-N} + \tfrac{t^2}{q} + \tfrac{t}{q-(N+3t+1)} + \tfrac{t+1}{2^k} \\
&\leq \tfrac{tN+3t(t+2)/2}{q-(N+3t+1)^2-N} + \tfrac{t^2}{q} + \tfrac{t+1}{2^k} = \mathcal{O}\left(\tfrac{t+N}{2^k}\right) .
\end{aligned}
$$

□

5 Multi-User Security of Other Fiat-Shamir Signatures

We recall that the core ideas in the security reduction of proving (multi-user) security of short Schnorr signatures are the following:

(1) Creating the known and the partially known set in the generic group model,
(2) Defining the event BRIDGE (resp. BRIDGE^N) and upper-bounding the probability of the event happens, and
(3) Programming the random oracle H when signing the message without the secret key.

Thus, we can follow the same technique in other Fiat-Shamir based group signature schemes and argue that such signatures are also provide k-bits of MU-UF-CMA-security in the generic group model and the programmable random oracle model.

Security Analysis of Chaum-Pederson Signatures. The Chaum-Pederson signature scheme [CP93] is obtained by applying the Fiat-Shamir transform to the Chaum-Pederson identification scheme and works as follows[5]. Let q be a prime and define $G = \langle g \rangle$ as the cyclic group of order q. In our presentation we assume that the message $m \in G$ as shown in Figure 4. One can easily extend the signature scheme to arbitrary message $m \in \{0,1\}^*$ using a collision-resistant hash function to map m to a group element $h(m) \in G$.

Then it is not difficult to see that the same reduction technique works for the Chaum-Pederson signature scheme as well. The key difference from the short Schnorr signatures is to sign a message without the secret key x when programming the random oracle as demonstrated in Figure 6.

[5] The Chaum-Pederson signature scheme is nearly identical to Katz-Wang signatures [KW03] except for the exclusion of the public key from the hash computation i.e., $e \leftarrow \mathsf{H}(pk, a, b, m)$ in [KW03].

```
Kg(1^k):
1: $sk \leftarrow_{\$} \mathbb{Z}_q$
2: $pk \leftarrow (q, g, g^{sk})$
3: return $(pk, sk)$
```

```
Sign(sk, m):
1: $r \leftarrow_{\$} \mathbb{Z}_q;\ z = m^{sk}$
2: $(a, b) \leftarrow (g^r, m^r)$
3: $e \leftarrow \mathsf{H}(m, z, a, b)$
4: $s \leftarrow r + sk \cdot e \mod q$
5: return $\sigma = (z, a, b, s)$
```

```
Vfy(pk, m, σ):
1: Compute $g^s, ag^{sk \cdot e}, m^s, bz^e$
2: if $g^s = ag^{sk \cdot e} \wedge m^s = bz^e$ then
3:     return 1
4: else return 0
```

Figure 4. The Chaum-Pederson Signature Scheme.

Since the reduction procedure can be naturally extended to the Chaum-Pederson signature scheme with the signing oracle above, we can argue that Theorem 6 holds. We remark that a Chaum-Pederson signature with k-bits of security has length $8k$ — each group element requires $2k$ bits to encode since $q \approx 2^{2k}$. Note that reducing the length of the hash output does not have any affect on Chaum-Pederson signature length. Thus, we assume that H is a random oracle with $2k$-bit outputs. We defer the proofs to Appendix B for the interested readers.

Theorem 6. *The Chaum-Pederson signature scheme is* $\left(t, N, q_{\mathsf{RO}}, q_{\mathsf{GO}}, q_{\mathsf{Sign}}, \mathcal{O}\left(\frac{t+N}{2^k}\right)\right)$-MU-UF-CMA *secure under the generic group model of prime order $q \approx 2^{2k}$ and the programmable random oracle model.*

Proof Sketch of Theorem 6: We follow the same security reduction as in Theorem 5 using the signing oracle in Figure 6 (left). One minor difference in the analysis is that the hash output is $2k$-bits instead of k-bits e.g., in Claim 6 the bound is $\Pr[\mathsf{BadQuery}^N] \leq 2^{-2k}$ instead of 2^{-k}. □

Security Analysis of Katz-Wang Signatures. The Katz-Wang signature scheme [KW03] is a double generator version of Schnorr signature scheme. Here, let $G = \langle g_1 \rangle = \langle g_2 \rangle$ be a cyclic group of prime order q with two generators g_1 and g_2. Here, the message space for m is arbitrary, i.e., $m \in \{0,1\}^*$.

```
Kg(1^k):
1: $sk \leftarrow_{\$} \mathbb{Z}_q$
2: $h_1 \leftarrow g_1^{sk}, h_2 \leftarrow g_2^{sk}$
3: $pk \leftarrow (g_1, g_2, h_1, h_2)$
4: return $(pk, sk)$
```

```
Sign(sk, m):
1: $r \leftarrow_{\$} \mathbb{Z}_q$
2: $a \leftarrow g_1^r, b \leftarrow g_2^r$
3: $e \leftarrow \mathsf{H}(pk, a, b, m)$
4: $s \leftarrow r + sk \cdot e \mod q$
5: return $\sigma = (s, e)$
```

```
Vfy(pk, m, σ):
1: $a' \leftarrow g_1^s h_1^{-e}, b' \leftarrow g_2^s h_2^{-e}$
2: if $e = \mathsf{H}(pk, a', b', m)$ then
3:     return 1
4: else return 0
```

Figure 5. The Katz-Wang Signature Scheme.

In this case, the signing oracle $\mathsf{Sign}(m)$ without the secret key x works as shown in Figure 6. We remark that this is almost the same as the one from the Schnorr signatures except the oracle computes for the two generators g_1 and g_2. We remark that the length of a regular Katz-Wang signature is $4k$

```
Sign(m) without secret key x
1 :  Pick $s, e, z$ randomly
2 :  Compute $\tau(g^s), \tau(m^s), \tau(g^{x\cdot e}), \tau(z^e)$
3 :  Compute $a = \tau(g^s \cdot g^{-xe})$,
4 :  Compute $b = \tau(m^s \cdot z^{-e})$
5 :  if $\mathsf{H}(m, z, a, b) \in$ prior query then
6 :      return $\perp$
7 :  else Program $\mathsf{H}(m, z, a, b) := e$
8 :      return $\sigma = (z, a, b, s)$
```

```
Sign(m) without secret key x
1 :  Pick $s, e$ randomly
2 :  Compute $\tau(g_i^s), \tau(g_i^{xe})$ for $i = 1, 2$
3 :  Compute $a = \tau(g_1^{s-xe}), b = \tau(g_2^{s-xe})$
4 :  if $\mathsf{H}(pk, a, b, m) \in$ prior query then
5 :      return $\perp$
6 :  else Program $\mathsf{H}(pk, a, b, m) := e$
7 :      return $\sigma = (s, e)$
```

Figure 6. The Signing Oracle without Secret Key in the Chaum-Pederson Scheme (Left) and the Katz-Wang Scheme (Right).

bits when $q \approx 2^{2k}$. Similar to Short Schnorr Signatures one can shorten the length of the hash output to k bits to obtain $3k$ bit signature. Essentially the same reduction can be used to demonstrate the multi-user security of (short) Katz-Wang signatures.

Theorem 7. *The (short) Katz-Wang signature scheme is* $\left(t, N, q_{\mathsf{RO}}, q_{\mathsf{GO}}, q_{\mathsf{Sign}}, \mathcal{O}\left(\frac{t+N}{2^k}\right)\right)$-MU-UF-CMA *secure under the generic group model of prime order* $q \approx 2^{2k}$ *and the programmable random oracle model.*

Proof Sketch of Theorem 7: We follow the same security reduction as in Theorem 5 using the signing oracle in as described in Figure 6 (right). In our reduction we assume that $\mathsf{DLog}_{g_1}(g_2)$ is publicly known. Thus, we can maintain our list $\mathcal{L}$ of tuples $(\mathfrak{y}, \boldsymbol{a}, b)$ s.t. $\mathfrak{y} = \tau\left(g_1^{\boldsymbol{x}\cdot\boldsymbol{a}+b}\right)$ where $\boldsymbol{x}[i]$ denotes the i^{th} secret key. □

Remark 1. Kiltz et al. [KMP16] showed that if the decisional Diffie-Hellman problem is (t, ϵ)-hard then an adversary who tries to forge one out of N (regular) Katz-Wang signatures running at most time t' can succeed with the probability ϵ' upper bounded by

$$\epsilon' \le t' \left(4 \cdot \frac{\epsilon}{t} + \frac{q_{\mathsf{Sign}}}{q} + \frac{1}{2^k}\right).$$

6 Non-Standard Generic Group Models

In our analysis we show that short Schnorr signatures provide k-bits of security in Shoup's original generic group model [Sho97]. Schnorr and Jakobsson's model [SJ00] previously established the security of short Schnorr signatures in their version of the generic group model using programmable random oracles. Similarly, Kiltz et al. [KMP16] proved that *regular* Schnorr signatures provide k-bits of security in the multi-user setting using a different version of the generic group model. In this section we review the different variants of the generic group model, and motivate why we chose to conduct our analysis in Shoup's original model [Sho97]. As a motivating example we consider a recent generic preprocessing attack on the squared Decisional Diffie-Hellman problem (sqDDH) [KM10] due to Corrigan-Gibbs and Kogan [CK18]. While it is straightforward to describe the attack in the standard generic group model [Sho97] it does appear to be not possible to describe the attack in non-standard variations of the generic group model.

6.1 The Generic Group Model based on Collisions

In the generic group model of Shoup the attacker is given a handle $\tau(h)$ for any group element that is the output of any generic group query. In the model of Schnorr and Jakobsson [SJ00] the attacker is not directly given a handle. In particular, if f_i denotes the output of the ith generic group query the attacker is simply informed whether or not f_i is a new group element or whether f_i collided with a prior query. The attacker may *indirectly* reference previously computed group elements by submitting a query $(a_1, \cdots, a_{i-1}) \in \mathbb{Z}_q^{i-1}$ to the generic group oracle. The attacker is then informed whether or not the group element $f_i := \prod_{j=1}^{i-1} f_j^{a_j}$ is new or not. If $f_i \in \{f_1, \ldots, f_{i-1}\}$ then the attacker is given the index $j < i$ of any group element such that $f_j = f_i$. The formal definition of a *generic algorithm*, as defined by Schnorr and Jakobsson [SJ00], is given in Definition 3.

Definition 3. [SJ00] *A (non-standard)* generic algorithm *is a sequence of t generic steps; for time $1 \leq t' < t$, the algorithm takes inputs as $f_1, \cdots, f_{t'} \in G$, and computes $f_i = \prod_{j=1}^{i-1} f_j^{a_j}$ for $i = t'+1, \cdots, t$, where $(a_1, \cdots, a_{i-1}) \in \mathbb{Z}_q^{i-1}$ depends arbitrarily on i, the non-group element and the set $\mathcal{CO}_{i-1} := \{(j,k) | f_j = f_k, 1 \leq j < k \leq i-1\}$ of previous "collisions" of group elements.*

Remark 2. While many natural attacks can be modeled using Definition 3, there are several limitations due to the lack of a direct handle on group elements. In Definition 3 the only way to obtain new group elements is by executing another generic step. By contrast, in Shoup's model the attacker can pick a binary string x at any time expecting that $x = \tau(h)$ for some group element $h \in G$ and submit x as the input to generic group oracles. Similarly, in Shoup's model the attacker can easily "mark" or partition the group elements G. In fact, this can be done *before* the attacker ever queries the generic group oracle(s). For example, we could define $G_0 = \{h : 0 = \tau(h) \mod 2\}$ and $G_1 = \{h : 1 = \tau(h) \mod 2\}$. Later when we are given the handle $\tau(h)$ it is trivial to test whether $h \in G_b$. Similarly, if we define $G_0 = \{h : \tau(h) = 0 \mod \sqrt{q}\}$ then it is easy to sample elements in G_0 in Shoup's model i.e., pick a random x such that $x = 0 \mod \sqrt{q}$ expecting that $x = \tau(h)$ for some $h \in G$.

Preprocessing Attacks on the sqDDH Problems. As an illustrative example of a generic attack which cannot be described in this non-standard generic group model we discuss the preprocessing attack on the sqDDH problem [KM10] proposed by Corrigan-Gibbs and Kogan [CK18]. The sqDDH problem requires to distinguish tuples of the form (g, g^x, g^y) from $(g, g^x, g^{(x^2)})$ for random $x, y \in \mathbb{Z}_q$. Corrigan-Gibbs and Kogan [CK18] introduced a sqDDH distinguisher $\mathcal{D}_{\text{sqDDH}}$ using preprocessing. The preprocessing attack generates a hint of size at most s (bits) in the offline phase after arbitrary interaction with the generic group oracles. In the online phase the attacker is given the pair $(\tau(h_1), \tau(h_2))$ and must guess whether the pair is a valid sqDDH pair. The attack of Corrigan-Gibbs and Kogan [CK18] runs in time t and achieves advantage ϵ provided that $st^2 = \Omega(q\epsilon^2)$ (see Theorem 8). Interestingly, this attack matches the lower bound for the regular DDH problem i.e., any preprocessing attack which achieves distinguishing advantage ϵ for DDH must have $st^2 = \tilde{\Omega}(q\epsilon^2)$.

Intuitively, the preprocessing phase takes advantage of the ability of the attacker to "mark" and/or "color" exponentially large subsets of pairs $(\mathfrak{u}_1, \mathfrak{u}_2) \in \mathbb{G}^2$ — a capability that the attacker does not have in Definition 3. In particular, the attack relies on several random functions $\mathsf{H}_\mathsf{m} : \mathbb{G}^2 \to \{1, \dots, t\}$ and $\mathsf{H}_\mathsf{c} : \mathbb{G}^2 \to \{1, \dots, s\}$ to "mark" and "color" vertices i.e., the set of marked vertices is $\mathcal{M} = \{(\mathfrak{u}_1, \mathfrak{u}_2) \in \mathbb{G}^2 : \mathsf{H}_\mathsf{m}(\mathfrak{u}_1, \mathfrak{u}_2) = 1\}$ and any marked node $(\mathfrak{u}_1, \mathfrak{u}_2) \in \mathcal{M}$ is assigned the color $\mathsf{H}_\mathsf{c}(\mathfrak{u}_1, \mathfrak{u}_2)$. Next Corrigan-Gibbs and Kogan define a random walk using a random function $f : \tau(G)^2 \to \mathbb{Z}_q$, i.e., starting at the node $(\tau(h_0), \tau(h_1)) \in \tau(G)^2$ we compute $\alpha \leftarrow f(\tau(h_0), \tau(h_1))$ and move to $(\tau(h_0^\alpha), \tau(h_1^{(\alpha^2)}) \in \tau(G)^2$. Letting $\mathcal{Y} = \{(\tau(g^x), \tau(g^{(x^2)})) : x \in \mathbb{Z}_q\} \subset \tau(G)^2$ denote the "yes" instance of the sqDDH problem it is easy to observe that walk that starts inside (resp. outside) $\mathcal{Y}$ remains inside (resp. outside) $\mathcal{Y}$. The random walk is used to select $\Omega\left(q/(3t^2)\right)$ marked nodes $\mathcal{T} \subseteq \mathcal{Y}$. Then for each color $c \in \{1, \dots, s\}$ the preprocessing algorithm computes the advice string $p_c \in \{0,1\}^{\log q}$ s.t.

$$p_c = \underset{p \in \{0,1\}^{\log q}}{\operatorname{argmax}} \sum_{\substack{(\mathfrak{m}, c_\mathfrak{m}) \in \mathcal{T}: \\ c_\mathfrak{m} = c}} H(p, \mathfrak{m}),$$

where $H : \{0,1\}^{\log q} \times \mathbb{G}^2 \to \{0,1\}$ is a random function. For completeness we include a description of the complete attack in Appendix A.

We now discuss some of the inherent challenges in modeling the attack why the non-standard generic group model cannot capture the sqDDH distinguishing attack illustrated above. When it comes to sampling a set of q^2/t "marked" points $\mathcal{M} := \{(\tau(y_0), \tau(y_1)) : \mathsf{H}_\mathsf{m}(\tau(y_0), \tau(y_1)) = 1\}$ we required an explicit handle $\tau(\cdot)$ in addition to the random hash function $\mathsf{H}_\mathsf{m} : \mathbb{G}^2 \to \{1, \dots, t\}$. Similarly, coloring each of the marked points using the random function H_m requires an explicit handle. Finally, we also require an explicit handle to compute each of the advice strings p_c using the hash function $H : \{0,1\}^{\log q} \times \mathbb{G}^2 \to \{0,1\}$.

6.2 The Generic Group Model using Incrementing Counters

While version of the generic group model used by Kiltz et al. [KMP16] is different from Schnorr and Jakobsson [SJ00], it is also not equivalent to the original generic group model of Shoup [Sho97]. In fact we show that any generic attack in the model of Kiltz et al. [KMP16] can we simulated within the model of Schnorr and Jakobsson [SJ00]. Thus, the model used by Kiltz et al. [KMP16] would also fail to capture attacks that require explicit handles on group elements such as the sqDDH preprocessing attack [CK18].

In the model of Kiltz et al. [KMP16] the generic group oracle maintains a global counter i which is incremented every time a new group element is produced. The generic group oracle also maintains a list of tuples which consists of the group element along with the corresponding counter, i.e., $(y, C_y) \in \mathbb{Z}_q \times \mathbb{N}$ where $\mathbb{N}$ denotes the set of positive integers. The counter C_y for each group element y is used as a "handle", e.g., if g and $h = g^x$ are the "public" group elements then the attacker is initially given the counters $C_g = 1$ and $C_h = 2$ and the next group element r that is generated will be assigned counter value $C_r = 3$.

Formally, the generic oracle works as following:

Oracle $\mathcal{O}_G$:
The oracle $\mathcal{O}_G$ takes input of two counters and output the resulting counter.

1. (Initialization) Let $g = g^1, h_1 = g^{x_1}, \ldots, h = g^{x_n}$ be the initial public elements. We add $(1, C_1 = 1)$ (the generator), and $(x_i, C_{x_i} = i + 1)$ for each $x_i \in \mathbb{Z}_q$ to our table. We also set our global counter $i = n + 1$ to count the number of group elements observed so far.
2. On input of two counters (C_a, C_b), the oracle searches the internal values (a, C_a) and (b, C_b), and computes $z = a + b \mod q$.
3. If the tuple (z, C_z) already is in the list, then output the counter C_z.
4. Otherwise, the counter i is increased by 1, the tuple $(z, C_z := i)$ is stored in the list, and the oracle outputs the counter C_z.

Consider a generic attack in the model above. We can translate this attack to an attack within model of Schnorr and Jakobsson [SJ00] by defining $\alpha_{C_a,C_b} \in \mathbb{Z}_q^*$ s.t. $\alpha_{C_a,C_b}[j] = 0$ for all $j \neq C_a, C_b$. If $C_a = C_b$ then $\alpha_{C_a,C_b}[C_a] = 2$ otherwise $\alpha_{C_a,C_b}[C_a] = \alpha_{C_a,C_b}[C_b] = 1$. If we let f_i be the group element corresponding to the counter i (following the notation of Schnorr and Jakobsson [SJ00]) then the response to the query α_{C_a,C_b} is $\prod_j f_j^{\alpha_{C_a,C_b}[j]} = f_{C_a} f_{C_b}$. Whenever our generic attack submits the query (C_a, C_b) to the oracle $\mathcal{O}_G$ we intercept this query and submit the query α_{C_a,C_b} to the Schnorr/ Jakobsson oracle. We are not given the output $f_{C_a} f_{C_b}$ directly, but we are informed whether or not a collision occurred (and where) which allows us to either increment the counter or (in the case of a collision) return the original counter.

Thus, any attack which cannot be described by the model of Schnorr and Jakobsson [SJ00] also cannot be described using the model of Kiltz et al. [KMP16].

Remark 3. Another limitation is that there is no `Inv` oracle in the Kiltz et al. [KMP16]. Thus, computing h^{-1} requires us to compute h^{q-1} using $\mathcal{O}(\log q)$ queries to the generic group model. Thus, to simulate an attacker who makes t queries to the `Mult`, `Inv` oracles we might require *up to* $t \log q$ queries if we only have the `Mult` oracle. Such a reduction increases running time by a factor of $\mathcal{O}(\log q)$ so we lose a factor of $\log q$ in the security reduction if their is an efficient algorithm to compute h^{-1} directly.

Acknowledgements

Seunghoon Lee was supported in part by the NSF Award CNS #1755708 and by the Center for Science of Information at Purdue. The opinions in this paper are those of the authors and do not necessarily reflect the position of the National Science Foundation and the Center for Science of Information at Purdue.

References

Ber15. Daniel J. Bernstein. Multi-user Schnorr security, revisited. Cryptology ePrint Archive, Report 2015/996, 2015. `http://eprint.iacr.org/2015/996`. 1

BL12. Daniel J. Bernstein and Tanja Lange. Two grumpy giants and a baby. Cryptology ePrint Archive, Report 2012/294, 2012. `http://eprint.iacr.org/2012/294`. 1.3

BLS04. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, September 2004. 1.3

BR93. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993. 1.3, 1.3

CK18. Henry Corrigan-Gibbs and Dmitry Kogan. The discrete-logarithm problem with preprocessing. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 415–447. Springer, Heidelberg, April / May 2018. 1, 1.3, 6, 6.1, 6.2, A, A, 8, 9

CKMS17. Sanjit Chatterjee, Neal Koblitz, Alfred Menezes, and Palash Sarkar. Another look at tightness ii: Practical issues in cryptography. pages 21–55, 12 2017. 1.3

CP93. David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 89–105. Springer, Heidelberg, August 1993. 5

Den02. Alexander W. Dent. Adapting the weaknesses of the random oracle model to the generic group model. In Yuliang Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 100–109. Springer, Heidelberg, December 2002. 1.3

DS19. David Derler and Daniel Slamanig. Key-homomorphic signatures: definitions and applications to multiparty signatures and non-interactive zero-knowledge. *Designs, Codes and Cryptography*, 87(6):1373–1413, Jun 2019. 1.3

Fis00. Marc Fischlin. A note on security proofs in the generic model. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 458–469. Springer, Heidelberg, December 2000. 1.3

FJS14. Nils Fleischhacker, Tibor Jager, and Dominique Schröder. On tight security proofs for Schnorr signatures. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 512–531. Springer, Heidelberg, December 2014. 1, 1.3

FST10. David Freeman, Michael Scott, and Edlyn Teske. A taxonomy of pairing-friendly elliptic curves. *Journal of Cryptology*, 23(2):224–280, April 2010. 1.3

GGH$^+$13. Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013. 1.3

GHS02. Pierrick Gaudry, Florian Hess, and Nigel P. Smart. Constructive and destructive facets of Weil descent on elliptic curves. *Journal of Cryptology*, 15(1):19–46, January 2002. 1.3

GMLS02. S. Galbraith, J. Malone-Lee, and N. P. Smart. Public key signatures in the multi-user setting. *Inf. Process. Lett.*, 83(5):263–266, September 2002. 1

GWZ15. Steven D. Galbraith, Ping Wang, and Fangguo Zhang. Computing elliptic curve discrete logarithms with improved baby-step giant-step algorithm. Cryptology ePrint Archive, Report 2015/605, 2015. `http://eprint.iacr.org/2015/605`. 1.3

JMV01. Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International Journal of Information Security*, 1(1):36–63, Aug 2001. 1.3

JS08. Tibor Jager and Jörg Schwenk. On the equivalence of generic group models. In Joonsang Baek, Feng Bao, Kefei Chen, and Xuejia Lai, editors, *ProvSec 2008*, volume 5324 of *LNCS*, pages 200–209. Springer, Heidelberg, October / November 2008. 1.3

KM07. Neal Koblitz and Alfred Menezes. Another look at generic groups, 2007. 1.3

KM10. Neal Koblitz and Alfred Menezes. Intractable problems in cryptography. Cryptology ePrint Archive, Report 2010/290, 2010. `http://eprint.iacr.org/2010/290`. 6, 6.1

KMP16. Eike Kiltz, Daniel Masny, and Jiaxin Pan. Optimal security proofs for signatures from identification schemes. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 33–61. Springer, Heidelberg, August 2016. 1, 1.3, 1, 6, 6.2, 6.2, 3

KW03. Jonathan Katz and Nan Wang. Efficiency improvements for signature schemes with tight security reductions. In Sushil Jajodia, Vijayalakshmi Atluri, and Trent Jaeger, editors, *ACM CCS 2003*, pages 155–164. ACM Press, October 2003. 5, 5

LM17. Bei Liang and Aikaterini Mitrokotsa. Fast and adaptively secure signatures in the random oracle model from indistinguishability obfuscation. Cryptology ePrint Archive, Report 2017/969, 2017. `http://eprint.iacr.org/2017/969`. 1.3

MVO91. Alfred Menezes, Scott A. Vanstone, and Tatsuaki Okamoto. Reducing elliptic curve logarithms to logarithms in a finite field. In *23rd ACM STOC*, pages 80–89. ACM Press, May 1991. 1.3

Nec94. V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Math Notes*, 55:165, 1994. 1.3

NPSW09. Gregory Neven, Nigel P. Smart, and Bogdan Warinschi. Hash function requirements for schnorr signatures. *Journal of Mathematical Cryptology*, 3, 05 2009. 1, 1.3

PH06. S. Pohlig and M. Hellman. An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance (corresp.). *IEEE Trans. Inf. Theor.*, 24(1):106–110, September 2006. 1.3

Pol78. John M. Pollard. Monte Carlo methods for index computation mod p. *Mathematics of Computation*, 32:918–924, 1978. 1.3

PS96. David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 387–398. Springer, Heidelberg, May 1996. 1, 1.3

RW14. Kim Ramchen and Brent Waters. Fully secure and fast signing from obfuscation. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 659–673. ACM Press, November 2014. 1.3

Sch90. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, August 1990. 1

Seu12. Yannick Seurin. On the exact security of Schnorr-type signatures in the random oracle model. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 554–571. Springer, Heidelberg, April 2012. 1, 1.3

Sha71. D. Shanks. Class number, a theory of factorization, and genera. In *1969 Number Theory Institute (Proc. Sympos. Pure Math., Vol. XX, State Univ. New York, Stony Brook, N.Y., 1969)*, pages 415–440, 1971. 1.3

Sho97. Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997. 1, 1.3, 1.3, 2.1, 6, 6.2

SJ00. Claus-Peter Schnorr and Markus Jakobsson. Security of signed ElGamal encryption. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 73–89. Springer, Heidelberg, December 2000. 1, 1, 6, 6.1, 3, 6.2, 6.2

Sma99. Nigel P. Smart. The discrete logarithm problem on elliptic curves of trace one. *Journal of Cryptology*, 12(3):193–196, June 1999. 1.3

SW14. Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014. 1.3

WZ11. Ping Wang and Fangguo Zhang. Computing elliptic curve discrete logarithms with the negation map. Cryptology ePrint Archive, Report 2011/008, 2011. `http://eprint.iacr.org/2011/008`. 1.3

A Preprocessing Attack on sqDDH

For completeness we include the description of the preprocessing attack of Corrigan-Gibbs and Kogan [CK18] on the sqDDH problem. The sqDDH problem requires to distinguish tuples of the form (g, g^x, g^y) from $(g, g^x, g^{(x^2)})$ for random $x, y \in \mathbb{Z}_q$. Corrigan-Gibbs and Kogan [CK18] introduced a sqDDH distinguisher $\mathcal{D}_{\mathrm{sqDDH}}$ using preprocessing. The preprocessing attack generates a hint of size at most s (bits) in the offline phase after arbitrary interaction with the generic group oracles. In the online phase the attacker is given the pair $(\tau(h_1), \tau(h_2))$ and must guess whether the pair is a valid sqDDH pair. The attack of Corrigan-Gibbs and Kogan [CK18] runs in time t and achieves advantage ϵ provided that $st^2 = \Omega(q\epsilon^2)$. Interestingly, this attack matches the lower bound for the regular DDH problem i.e., any preprocessing attack which achieves distinguishing advantage ϵ for DDH must have $st^2 = \tilde{\Omega}(q\epsilon^2)$.

Consider a cyclic group $G = \langle g \rangle$ of prime order $q \approx 2^{2k}$ and $\tau : G \to \mathbb{G}$ a random injective encoding map from G from the set of bit strings $\mathbb{G}$. In this algorithm we consider the image of the group G through the map τ, i.e., $\tau(G) \subset \mathbb{G}$. Corrigan-Gibbs and Kogan define the subset of vertices $\mathcal{Y} = \{(\tau(g^x), \tau(g^{(x^2)})) : x \in \mathbb{Z}_q\} \subset \tau(G)^2$ that correspond to "yes" instance of the sqDDH problem.

Algorithm $\mathcal{D}_{\text{sqDDH}}$:
The algorithm takes as input $s, t \in \mathbb{Z}^+ := \{1, 2, 3, \ldots\}$, a sqDDH challenge $(\tau(h_0), \tau(h_1)) \in \tau(G)^2$ and outputs "1" if it is a valid sqDDH pair and outputs "0" otherwise.

1. *Defining a walk, marking and coloring.*
   (a) To define a walk, use a random function $f : \tau(G)^2 \to \mathbb{Z}_q$. Given a point $(\tau(h_0), \tau(h_1)) \in \tau(G)^2$, the walk computes $\alpha \leftarrow f(\tau(h_0), \tau(h_1))$ and moves to $(\tau(h_0^{\alpha}), \tau(h_1^{(\alpha^2)}) \in \tau(G)^2$. It is easy to observe that if the walk starts in $\mathcal{Y}$ (resp. not in $\mathcal{Y}$) then the walk remains inside of $\mathcal{Y}$ (resp. outside of $\mathcal{Y}$.)
   (b) There are q^2 total vertices of the form $(\tau(g^i), \tau(g^j)) \in \tau(G)^2, 0 \leq i, j \leq q-1$ in the graph. Among them, Corrigan-Gibbs and Kogan use a random function $\mathsf{H}_\mathsf{m} : \mathbb{G}^2 \to \{1, \ldots, t\}$ to "mark" points independently at random with probability $1/t$, e.g., the point $(\mathfrak{a}, \mathfrak{b}) \in \mathbb{G}^2$ is marked if and only if $\mathsf{H}_\mathsf{m}(\mathfrak{a}, \mathfrak{b}) = 1$. Similarly, marked points are assigned to one of s different "colors" using a separate random function $\mathsf{H}_\mathsf{c} : \mathbb{G}^2 \to \{1, \ldots, s\}$. Denote the set of marked points by $\mathcal{M} \subset \mathbb{G}^2$.
2. *Preprocessing phase.*
   (a) Choose $q/(3t^2)$ random points in $\mathcal{Y}$. From each point, take $2t$ steps of the walk on $\tau(G)^2$ that the function f defines. Halt the walk when reach a marked point $\mathfrak{m} \in \mathcal{M}$. If the walk hits the point $\mathfrak{m}$, store the point $\mathfrak{m}$ along with its color $c_\mathfrak{m}$ in a table. We call this table $\mathcal{T}$.
   (b) Group the endpoints of the walks by color. For each color $c \in \{1, \ldots, s\}$, find the prefix string $p_c \in \{0,1\}^{\log q}$ such that
$$p_c = \operatorname*{argmax}_{p \in \{0,1\}^{\log q}} \sum_{\substack{(\mathfrak{m}, c_\mathfrak{m}) \in \mathcal{T}: \\ c_\mathfrak{m} = c}} H(p, \mathfrak{m}),$$
   where $H : \{0,1\}^{\log q} \times \mathbb{G}^2 \to \{0,1\}$ is a random function.
   (c) Store the prefix strings $(p_1, \ldots, p_s)$ as the distinguisher's advice.
3. *Online phase.*
   (a) Given a sqDDH challenge $(\tau(h_0), \tau(h_1)) \in \tau(G)^2$, perform at most $10t$ steps of the walk on $\tau(G)^2$ as defined before.

- When the walk hits a marked point $\mathfrak{m} \in \mathcal{M}$ of color c, then return the value $H(p_c, \mathfrak{m})$.
- If the walk never hits a marked point, then return "0" or "1" with probability $1/2$ each.

Theorem 8 shows that the preprocessing attack by Corrigan-Gibbs and Kogan [CK18] illustrated above gives that $st^2 = \Omega(\epsilon^2 q)$ which matches the lower bound for the regular DDH problem. Corrigan-Gibbs and Kogan also remarked that if we consider the pseudo-random generator $P(x) := (\tau(g^x), \tau(g^{(x^2)}))$ that maps $\mathbb{Z}_q$ to $\mathbb{G}^2$, then with the generic algorithms with preprocessing it becomes significantly easier to distinguish this PRG from random than it is to compute discrete logs.

Theorem 8. [CK18] *There is a sqDDH distinguisher with preprocessing that makes use of a random function, uses $\widetilde{\mathcal{O}}(s)$ bits of group-specific advice, runs in time $\widetilde{\mathcal{O}}(t)$, and achieves distinguishing advantage ϵ whenever $st^2 = \Omega(\epsilon^2 q)$.*

Another significance of this preprocessing attack comes to arise when compared to the generic lower bound for the discrete-log problems (Theorem 9). Consider a simple sqDDH distinguisher with input $(\tau(h_0), \tau(h_1)) \in \mathbb{G}^2$, computes the discrete-log $x = \log_g(h_0)$ and checks whether $\tau(h_1) = \tau(g^{(x^2)}) \in \mathbb{G}$. Theorem 9 indicates that this simple attack allows the parameter setting $s = t = 1/\epsilon = q^{1/4}$ since $st^2 = \widetilde{\Omega}(\epsilon q)$. In contrast, the distinguisher in Theorem 8 allows the better running time and advice complexity $s = t = 1/\epsilon = q^{1/5}$ as the preprocessing attack has $st^2 = \Omega(\epsilon^2 q)$.

Theorem 9. [CK18] *Let G be a cyclic group of prime order q. Let $(\mathcal{A}_0, \mathcal{A}_1)$ be a pair of generic algorithms for G on $\mathbb{G}$, such that $\mathcal{A}_0$ outputs an s-bit state, $\mathcal{A}_1$ makes at most t oracle queries, and*

$$\Pr_{\tau, x, \mathcal{A}_1}\left[\mathcal{A}_1^{\mathsf{GO}}\left(\mathcal{A}_0^{\mathsf{GO}}(\tau(1)), \tau(x)\right) = x\right] \geq \epsilon,$$

where the probability is taken over the uniformly random choice of the encoding function $\tau : G \to \mathbb{G}$, the instance $x \in \mathbb{Z}_q$, and the coins of $\mathcal{A}_1$. Then $st^2 = \widetilde{\Omega}(\epsilon q)$.

B Missing Proofs

Reminder of Lemma 1. *The probability the attacker running in time t wins the generic discrete-log game (even with access to the restricted* $\mathtt{DLog}$ *oracle) is at most*

$$\Pr\left[\mathsf{DLogChal}_{\mathcal{A}}^{\mathsf{GO}}(k) = 1\right] \leq \frac{3(t+1)t + 3}{q - (3t+2)^2}$$

where q is the order of the group G.

Proof of Lemma 1: Without loss of generality, we can assume that for every generic group query involving a "fresh" $\tau(y)$ that the attacker queries $\mathtt{DLog}_g(\tau(y))$

before making the query (We say WLOG because the attacker can always ignore the result.) Thus, we will build up the known sets $\mathcal{K}$ (initially contains $(\tau(g), 0, 1)$) and the partially known set $\mathcal{PK}_x$ (initially contains $(\tau(h), 1, 0)$) as described in Section 1.2. Recall that we have the following observations for those sets:

– Initially, $\mathcal{K}$ contains $(\tau(g), 0, 1)$ and $\mathcal{PK}_x$ contains $(\tau(h) = \tau(g^x), 1, 0)$.
– If $(\mathfrak{a}_1, 0, c_1), (\mathfrak{a}_2, 0, c_2) \in \mathcal{K}$, then $(\texttt{Mult}(\mathfrak{a}_1, \mathfrak{a}_2), 0, c_1 + c_2)$ can be added to $\mathcal{K}$.
– If $(\mathfrak{a}_1, a_1, b_1), (\mathfrak{a}_2, a_2, b_2) \in \mathcal{PK}_x$, then $(\texttt{Mult}(\mathfrak{a}_1, \mathfrak{a}_2), a_1 + a_2, b_1 + b_2)$ can be added to $\mathcal{PK}_x$ except for a special case when $a_1 + a_2 = 0$ in which case the tuple $(\texttt{Mult}(\mathfrak{a}_1, \mathfrak{a}_2), 0, b_1 + b_2)$ should be added to $\mathcal{K}$ instead.
– If $(\mathfrak{a}, 0, c) \in \mathcal{K}$ and $(\mathfrak{b}, a, b) \in \mathcal{PK}_x$, then $(\texttt{Mult}(\mathfrak{a}, \mathfrak{b}), a, b + c)$ can be added to $\mathcal{PK}_x$.
– If $(\mathfrak{a}, a, b) \in \mathcal{K}$ (resp. $\mathcal{PK}_x$), then $(\texttt{Inv}(\mathfrak{a}), -a, -b)$ can be added to $\mathcal{K}$ (resp. $\mathcal{PK}_x$.)

Now consider the event BRIDGE which is the event that one of the following occurs when querying $\texttt{Mult}$, $\texttt{Inv}$ or $\texttt{DLog}$:

(1) The output of a query which could be added to $\mathcal{K}$ is already found in $\mathcal{PK}_x$, and
(2) The output of a query which could be added to $\mathcal{PK}_x$ is already found in $\mathcal{K}$.

Intuitively, the attacker wants the event BRIDGE to occur, e.g., in either case we have found an element $\tau(y)$ which can be written in two ways; (1) $\tau(y) = \tau(g^r)$ for some known $r \in \mathbb{Z}_q$ and (2) $\tau(y) = \tau(g^{ax+b})$ for known $a, b \in \mathbb{Z}_q$ with $a \neq 0$. This allows us to solve for $x = (r - b)a^{-1}$.

If the event BRIDGE does not occur then after all queries have finished the attacker can still view $x \notin \mathcal{K}$ as an element yet to be sampled from a uniform distribution over a set of size at least $q - |\mathcal{K}| \times |\mathcal{PK}_x|$. To see this note that each pair of distinct elements $\tau(g^{ax+b})$ in $\mathcal{PK}_x$ and $\tau(g^r)$ in $\mathcal{K}$ eliminates at most one possible value of x, i.e., since $\tau(g^{ax+b})$ and $\tau(g^r)$ are distinct we have $x \neq (r - b)a^{-1}$. We further note that $|\mathcal{K}| + |\mathcal{PK}_x| \leq 3t + 1$ since each query to the generic group model adds at most 3 elements to $\mathcal{K} \cup \mathcal{PK}_x$ — equality holds when both inputs to $\texttt{Mult}(\cdot, \cdot)$ are fresh. The probability the attacker guesses x correctly is at most

$$\Pr\left[\mathsf{DLogChal}^{\texttt{GO}}_{\mathcal{A}}(k) = 1 \,\middle|\, \overline{\mathsf{BRIDGE}}\right] \leq \frac{1}{q - |\mathcal{K}| \times |\mathcal{PK}_x|} < \frac{1}{q - (3t + 2)^2}\ .$$

To compute the probability that BRIDGE occurs we use our prior observation that the combined size of $|\mathcal{K}|$ and $|\mathcal{PK}_x|$ is at most $(3t + 2)$. Consider the ith query to the generic group oracle and let $\overline{\mathsf{BRIDGE}}_{<i}$ be the event that we have not yet seen a bridge query. Conditioning on this event we can view x as a yet to be selected value uniformly sampled from a set of size at least $q - (3t + 2)^2$. Let $\mathfrak{y}_i$ be the output of query i and let $(\mathfrak{y}_i, a_i, b_i)$ be the tuple that is added to $\mathcal{PK}_x \cup \mathcal{K}$. For each tuple $(\mathfrak{y}, a, b)$ in $\mathcal{K} \cup \mathcal{PK}_x$ the probability that $x = (b_i - b)(a - a_i)^{-1}$ is at most $\frac{1}{q - (3t+2)^2}$. Union bounding over all $(\leq 3i + 2)$ such tuples in $\mathcal{PK}_x \cup \mathcal{K}$ the probability of the event B_i that the ith query bridges is at most

$$\Pr\left[B_i \;\middle|\; \overline{\mathsf{BRIDGE}_{<i}}\right] \le \frac{3i+2}{q-(3t+2)^2} \;.$$

Therefore, the probability of BRIDGE is upper bounded by

$$\begin{aligned}\Pr[\mathsf{BRIDGE}] &= \sum_{i\le t} \Pr\left[B_i \;\middle|\; \overline{\mathsf{BRIDGE}_{<i}}\right] \\ &= \sum_{i\le t} \frac{3i+2}{q-(3t+2)^2} \\ &= \frac{3(t+1)t+2}{q-(3t+2)^2} \;.\end{aligned}$$

Thus, the probability the attacker succeeds is upper bounded by[6]

$$\begin{aligned}\Pr\left[\mathsf{DLogChal}^{\mathtt{GO}}_{\mathcal{A}}(k)=1\right] &\le \Pr\left[\mathsf{BRIDGE}\right] + \Pr\left[\mathsf{DLogChal}^{\mathtt{GO}}_{\mathcal{A}}(k)=1 \;\middle|\; \overline{\mathsf{BRIDGE}}\right] \\ &\le \frac{3(t+1)t+3}{q-(3t+2)^2} \;.\end{aligned}$$

i.e., we would need $t=\mathcal{O}\left(\sqrt{q}\right)$ queries to succeed with constant probability. □

Reminder of Claim 1. $\Pr[\mathsf{FailtoSign}] \le q_{\mathsf{Sign}} \times \frac{q_{\mathtt{RO}}+q_{\mathsf{Sign}}}{q}$.

Proof of Claim 1: Since, $I=\mathtt{Pow}(\tau(g),r)$ represents a fresh/randomly selected group element we can use union bounds to upper bound the probability that this failure event ever occurs as

$$\Pr[\mathsf{FailtoSign}] \le q_{\mathsf{Sign}} \times \frac{q_{\mathtt{RO}}+q_{\mathsf{Sign}}}{q}$$

where q_{Sign} is the total number of queries to the signing oracle and $q_{\mathtt{RO}}$ is the total number of queries to the random oracle. □

Reminder of Claim 2. $\Pr[\mathsf{FailtoFind}(I_\sigma)] \le \frac{q_{\mathtt{RO}}+q_{\mathsf{Sign}}}{q-|\mathcal{L}|} + \frac{1}{2^k}$.

Proof of Claim 2: If $I_\sigma \notin \mathcal{L}$ we can view $I_\sigma = \tau(g^s \cdot g^{-xe})$ as a uniformly random binary string from a set of size at least $q-|\mathcal{L}|$ which had not yet been selected at the time $\mathcal{A}_{\mathsf{sig}}$ output σ. Thus, the probability that the query $\mathsf{H}(I_\sigma||m)$ was previously recorded is at most $(q_{\mathtt{RO}}+q_{\mathsf{Sign}})/\left(q-|\mathcal{L}|\right)$. Observe that if the query $\mathsf{H}(I_\sigma||m)$ *was not* previously recorded then the probability of a successful forgery $\mathsf{H}(I_\sigma||m)=e$ is *at most* 2^{-k} since we can view $\mathsf{H}(I_\sigma||m)$ as a uniformly random k-bit string. Hence,

$$\Pr[\mathsf{FailtoFind}(I_\sigma)] \le \frac{q_{\mathtt{RO}}+q_{\mathsf{Sign}}}{q-|\mathcal{L}|} + \frac{1}{2^k}.$$

[6] To see this note that for the events A and B we have that $\Pr[A] = \Pr[A \wedge B] + \Pr[A \wedge \overline{B}] \le \Pr[B] + \Pr[A|\overline{B}]$.

□

Reminder of Claim 3. $\Pr[\mathsf{BadQuery}] \leq \frac{q_{\mathtt{RO}}}{2^k}$.

Proof of Claim 3: We first note that by construction we ensure that the triple (I, a, b) will always be recorded in $\mathcal{K}$ or $\mathcal{PK}_x$ before a query of the form $\mathsf{H}(I||m)$ is ever issued — if I is new then we call $\mathtt{DLog}_g(I)$ before querying the random oracle. We call a random oracle query $x = I||m$ "bad" if $\mathsf{H}(x) = a$ where the triple (I, a, b) has already been recorded. Recall that if there were two recorded triples (I, a, b) and (I, c, d) then our algorithm would have already found x. Thus, the probability each individual query is "bad" is at most 2^{-k} and we can use union bounds to upper bound the probability of any "bad" query as

$$\Pr[\mathsf{BadQuery}] \leq \frac{q_{\mathtt{RO}}}{2^k}.$$

□